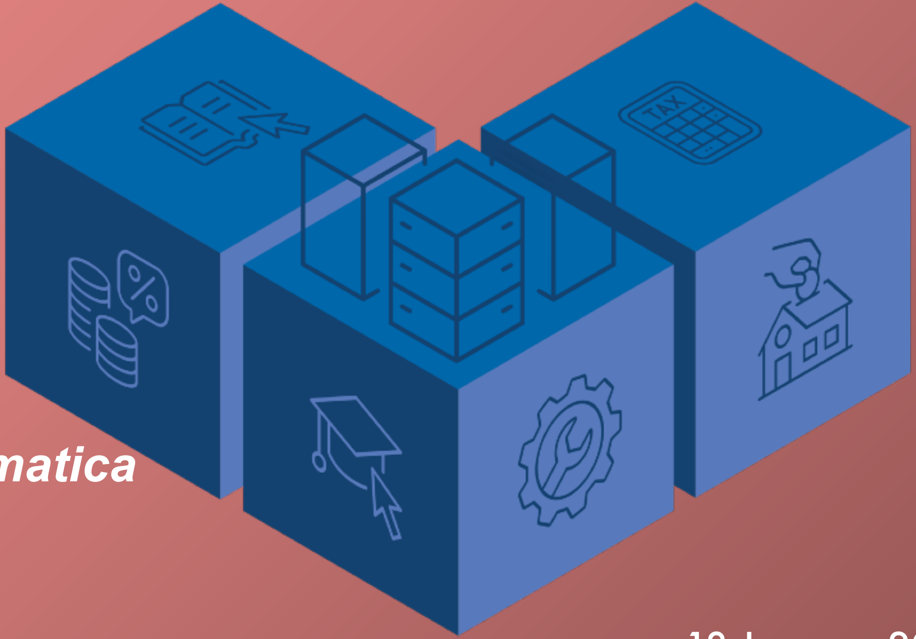
Software Engineering with COBOL and Mainframe: how *special* is that?



Jurgen Vinju

TU Eindhoven
Swat.engineering BV
Centrum Wiskunde & Informatica
and VERSEN

Future of
COBOL & Mainframe

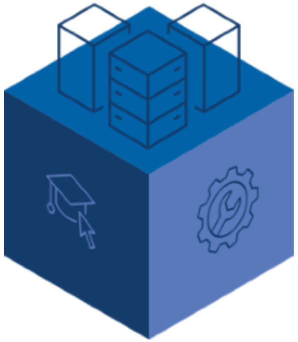
18 January 2024

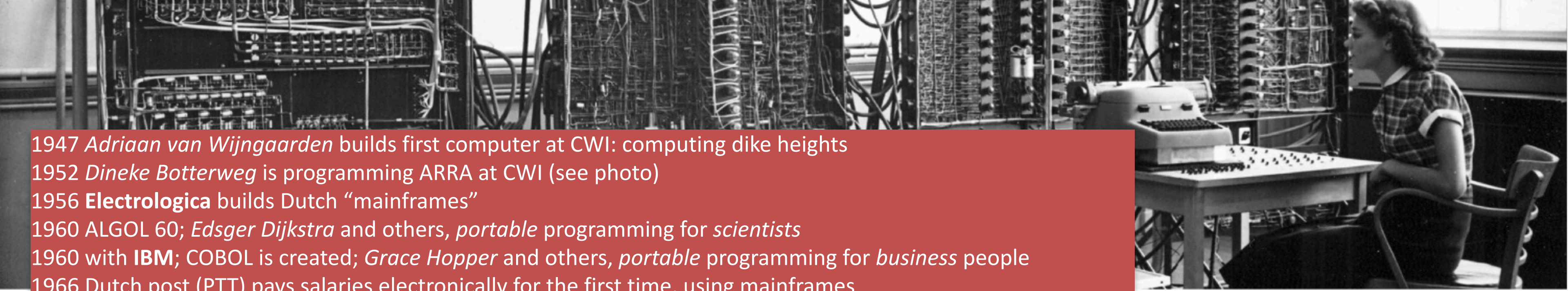
The Future of COBOL and Mainframe

➡What do we have in common?

➡What makes us special?

➡Where are we going?



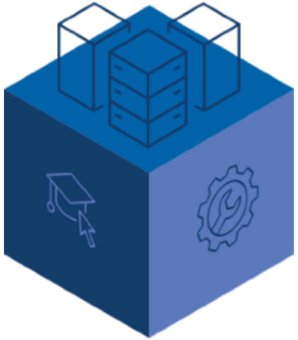


1947 *Adriaan van Wijngaarden* builds first computer at CWI: computing dike heights
1952 *Dineke Botterweg* is programming ARRA at CWI (see photo)
1956 **Electrologica** builds Dutch "mainframes"
1960 ALGOL 60; *Edsger Dijkstra* and others, *portable* programming for *scientists*
1960 with **IBM**; COBOL is created; *Grace Hopper* and others, *portable* programming for *business* people
1966 Dutch post (PTT) pays salaries electronically for the first time, using mainframes
1968 **Philips Computer Industrie** takes over Electrologica
1968 First NATO conference on Software Engineerig in Garnisch GE: The **SOFTWARE CRISIS** exists…
1968 "Goto considered harmful" by *Edsger Dijkstra*. *Programming languages make the difference*.
1974 COBOL-74 ANSI standard published
1974 CWI *Paul Klint* introduces Unix in Europe, on a **tape**
1976 First NASA Conference on Software Engineering
1979 Wim Ebbinkhuijsen starts work on COBOL 85 @ ISO for Philips Computer Industrie
1982 First curriculum Computer Science ("Informatica") appears in Dutch universities
1984 First debit card payment in a gas station in The Hague
1988 CWI *Piet Beertema* connects the "internet" to Europe
1990 Switzerland: *Tim Berners Lee* invents **HTTP** and the "World Wide Web"
1993 *Peter de Jager* (SA) calls out the **Y2K problem**
1994 Hack-Tic & gemeente Amsterdam launch "DDS: De Digitale Stad"
1995 Postbank introduces "telebankieren" for the public
1995 Belastingdienst introduces "elektronische aangifteprogramma"
2000 Y2K is an anticlimax for the public and a triumph for the insiders
2003 CWI, UvA, VU and HvA start a joint **master's** in "Software Engineering"
2007 Belgium goes **IBAN**
2014 The Netherlands go **IBAN**

(disclaimer: year numbers found after a night of intense googling)

For a field of engineering,
we share an exciting
(and relatively short) history…

… Netherlands is often
ahead of the pack!

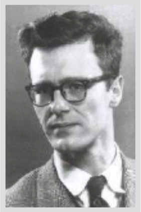


COBOL: an industrial design “form follows function”

Grace Hopper
[wikipedia]

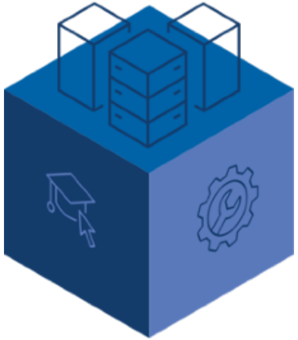
“[...] Manipulating symbols was fine for **mathematicians** but it was no good for **[administrators]**
[...] **I** decided [administrators] **ought to be able to write their programs in English**, [...]
[...] That was the beginning of **COBOL**
[Now they] could say **'SUBTRACT INCOMETAX FROM PAY’** [....]”

Edsger Dijkstra
[wikipedia]
“COBOL is bad”



```
ALGOL:    `PAY := PAY - INCOMETAX`
LISP:     `(setq PAY (- PAY INCOMETAX))`
COBOL:    `SUBTRACT INCOMETAX FROM PAY`
```

COBOL was invented to onboard more “non-mathy” people.
That was a huge success, compared to other languages.
It was the **gateway** to buying more mainframes.



What is [COBOL] Software Engineering?

[COBOL] [*Software*] Engineering is the application of *scientific*, **economic**, **social**, and **practical** knowledge in order to **invent**, design, build, **maintain**, and *improve* [COBOL] [*Software*].

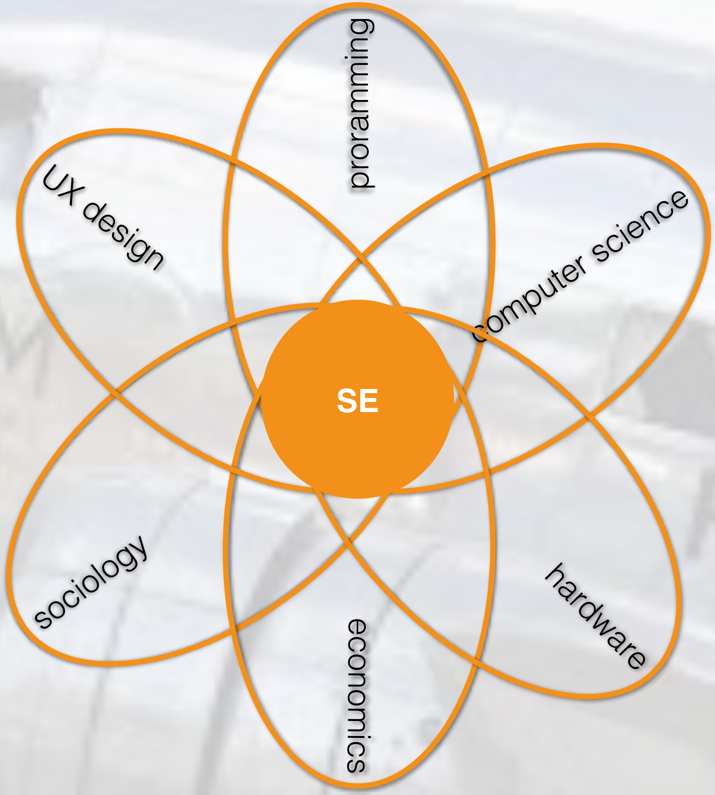
So, what do we know about [COBOL] software engineering?

First, we know more about what we don’t know…

Future of
COBOL & Mainframe

Wicked Problem

security
privacy
robustness
maintainability
usability
efficiency
energy
scalability
availability
flexibility
cost
…

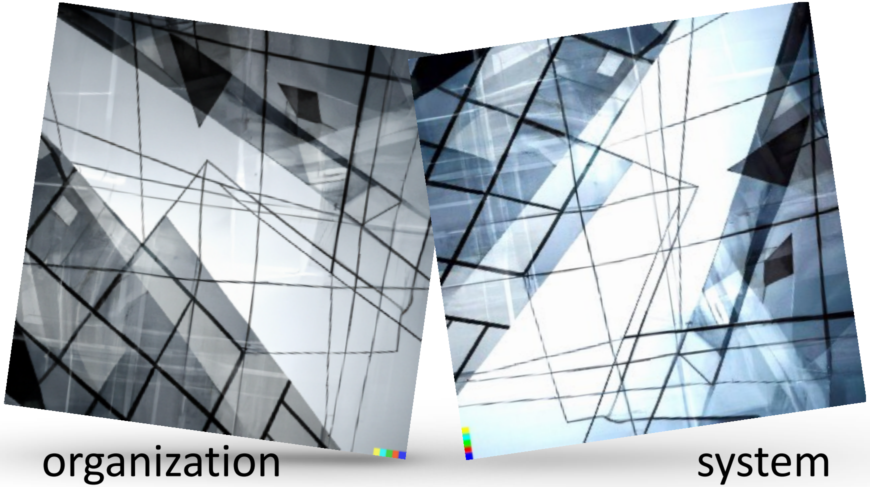


requirements
architecture
design
testing
construction
evolution
configuration
deployment

There are “Laws” of Software Engineering

Conway’s Law:

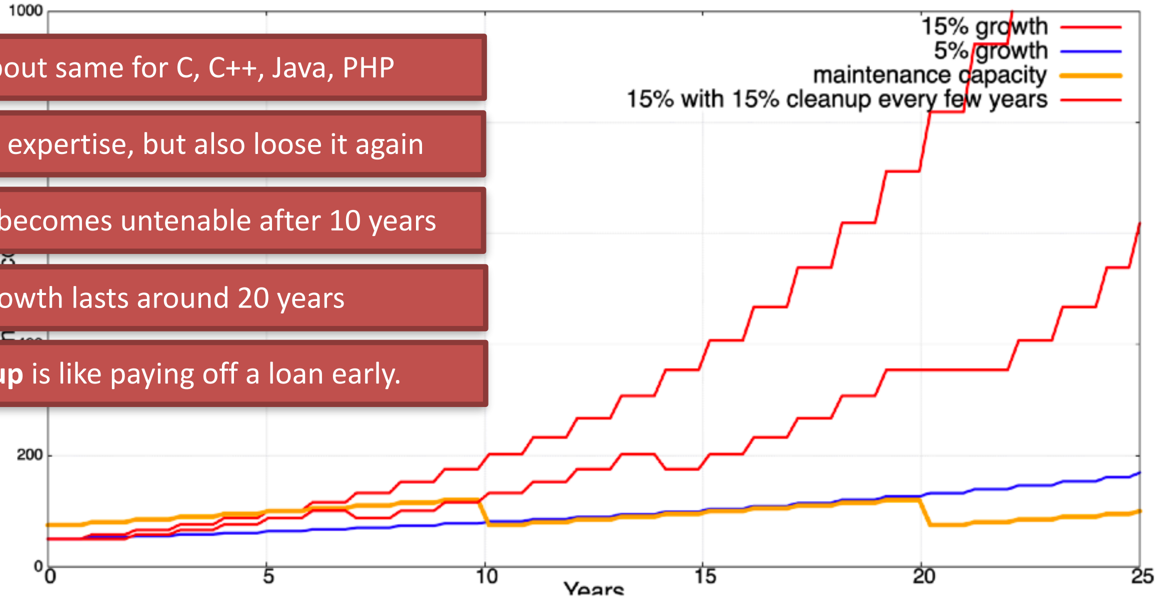
Communication patterns, collaboration, and relationships within **organizations** will *shape* the **architecture** and **design** of the software **systems** they develop [, *and vice versa*].



More “Laws” of Software Engineering

Lehman’s Laws of Software Evolution:

1. Software always has to change; *because it can*.
2. Software always grows [more complex]; *because there is no opposite force*



It’s the about same for C, C++, Java, PHP

Teams build expertise, but also loose it again

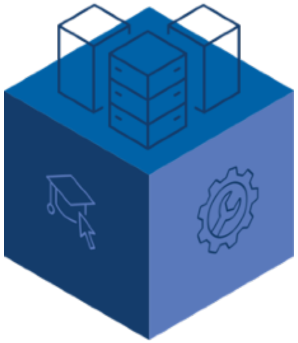
15% growth becomes untenable after 10 years

5% growth lasts around 20 years

Cleaning up is like paying off a loan early.

15% yearly code growth
is “typical” (anecdotal evidence)

code growth is like % interest:
cumulative growth is cumulative
costs and risks



The “Laws” of Software Engineering

Vinju’s Law of Software Contexts:

In Software Engineering context (sector, domain, legacy, culture) dominates software design decisions.

General theory may tell us *where* all the trade-offs may be, context knowledge tells us *how to balance* them.

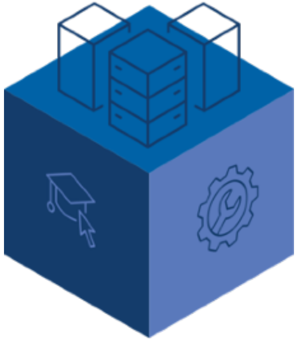
That’s why you can’t find answers to your software questions in a book.

That’s why your new employees, masters and bachelors interns, “don’t know anything” and “can’t fix anything”.

That’s why you end up depending on the “heroes” in your organization. Bus factor=1

That’s why outsourcing maintenance of essential software systems is *very* risky.

This is why you’ll find special **ownership** of, and **pride** in, the systems of **your** organization



Software complexity

Often more "intricate", like a knitted sweater, than "complex", like Fermat's last theorem.

Together, those "laws" explain software complexity.
Software complexity explains low productivity or inflexibility.

- Complex "*algorithms*" are often well-isolated and well-understood
  - *Parametrized computation of monthly mortgage payments*
  - **Specified** requirements, and theory on **performance** characteristics.

- Intricate "*processes*" escape architectural boundaries, **emergent** effects:
  - *A new business customer is onboarded into all relevant systems.*
  - There is no well-defined idea of what to expect, and we don't know what to expect of performance either.
  - Responsibilities are spread over asynchronously running independent components that still depend on each other.

Paradoxically, the seemingly easy parts are hard and the provenly hard parts are "easy" in software engineering :-)

In Dutch: "wel ingewikkeld maar niet zo moeilijk."

Why better programming languages (don't) matter

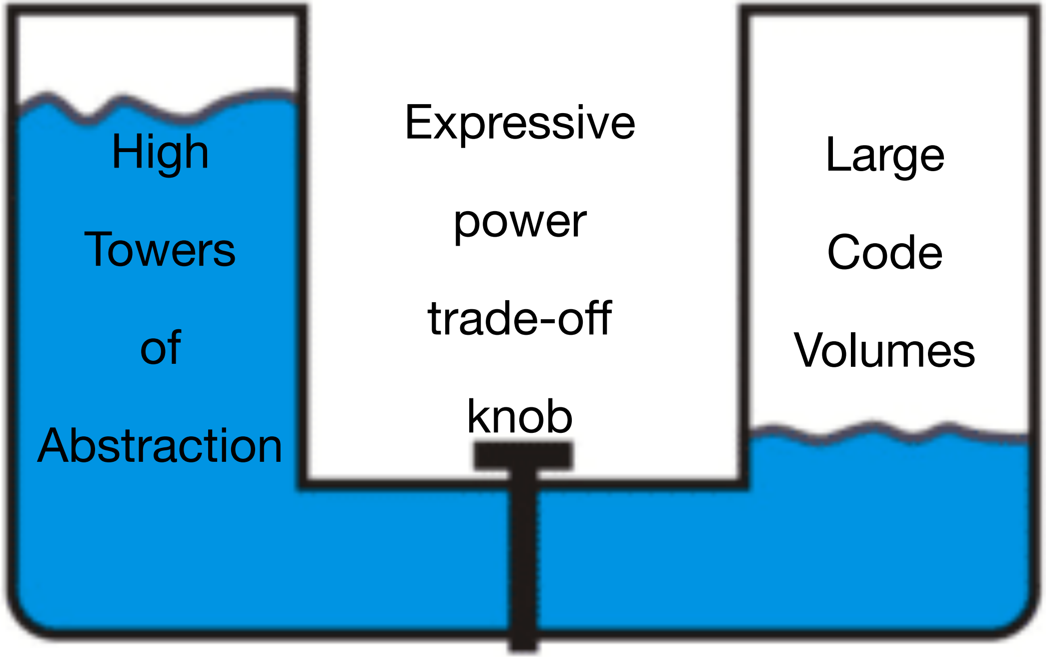
C++

Scala

Python

COBOL 2002

Java



COBOL 74

C

FORTRAN

Pascal, Modula 2, **COBOL 85**

it's cultural.

A rough estimate; in the Turing room, *right now*, we are responsible for 1,500 million (1.5 billion) lines of COBOL code. Wow!

COBOL and Mainframe; what’s (not) so different?

✓ COBOL has evolved to a modern programming language

✓ Mainframes are supercomputers

✓ COBOL has modern IDEs (code editors and browsers)

✓ And it works! Y2K, EURO, IBAN, SEPA, AVG/GDPR, negative interest, batch-to-online, you managed it all!

✓ So, what’s the deal?

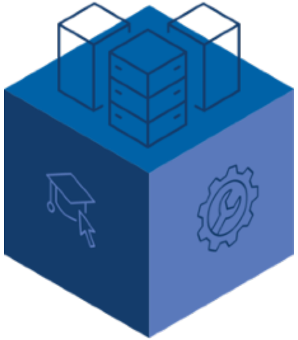
➡ COBOL systems are hugely successful; they enable the Netherlands’ economy and beyond like nothing else.

➡ they’ve **grown [complex] for many decades;** Because they were there **first** they trump everything else.

➡ they are **enabling economic infrastructure**, and so society **needs** and demands **100% availability**.

➡ The Netherlands has **not educated** enough COBOL programmers (yet); instead, we taught C, Java, C#, Prolog, Pascal, PHP, Javascript, Typescript, Python, Scala, C++, Haskell, and even Rascal.

➡ **Inmaintainability**: inability to *respond in time* with correct code changes to *changing requirements*



The Future of COBOL and Mainframe

Different directions we see and hear about today:

1. **Rationalization** and **simplification** of COBOL assets;

2. Incremental replacement of **components** {in other languages, on other platforms};

3. Big bang **semi-automated renovation**; let's get it over with.

4. Sticking with the **original plan**.

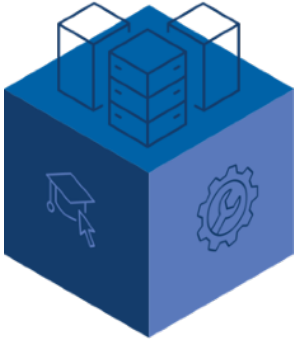
All those require a **deep understanding** of **COBOL assets**:

1. Because they (should) reflect your organization's rules, values, and structure;

2. Because incremental maintenance requires decomposition into modules

3. Because to replace something, you need to know what that "something" is.

4. Because changing a system with availability guarantees, requires predictability.



Personnel and finance are not my cup of tea, but go find out from the other speakers… wow!

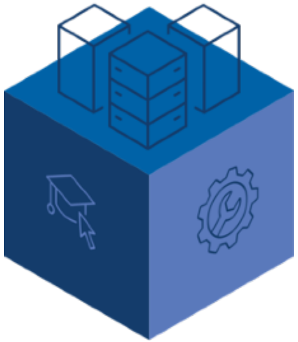
Understanding COBOL assets; how?

1. **`personnel`** : we need fresh expertise; how do we teach it, grow it, nurture it, keep it around?
2. **`finance`** : budget for continuous maintenance, and buying **quality** instead of **volume**
3. **`technology`**: high-tech SME’s and research institutes enough: co-create, co-maintain, co-innovate

From CWI and Swat.engineering our message is **technological** and **cultural**:

“*source code is data too*”.

What does that bring you?



Source code is data too. So what?

[COBOL] source code mixes the “**what**” with the “**where**” and “**how**” in code:

- “what” is the legal rule for tax-deductible traveling costs?
- “where” and “how” is that rule implemented in UI, database, server, etc. at the tax administration?
- the design synthesis of these aspects in code is called “software architecture” and “programming”
- note that the “why” is lost in this Bermuda triangle of design; that becomes “tacit” knowledge.

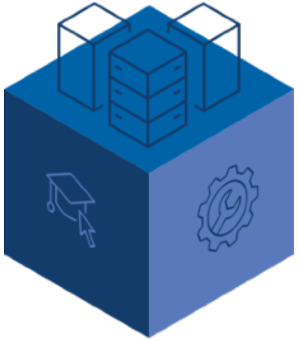
[COBOL] source code is too long to read, let alone understand.

Can this tangled knot of dependencies be untangled?

earch

Can we prevent this tangling for the future?

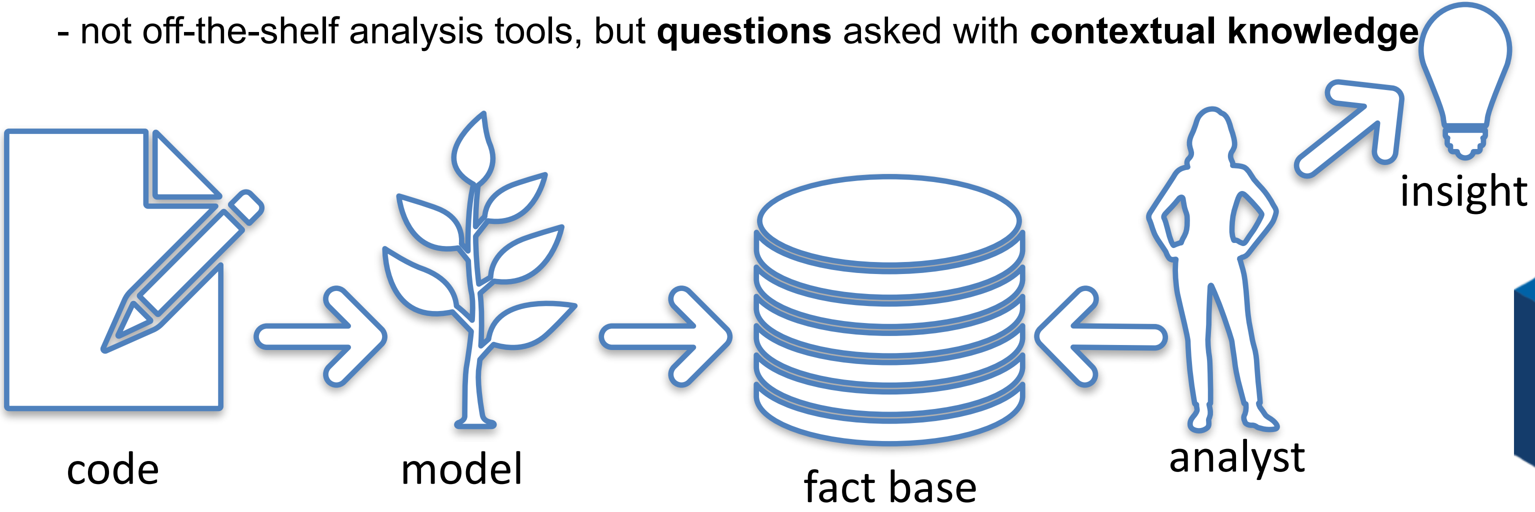
(both) by treating the code as data?

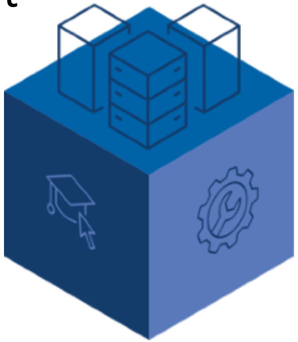


COBOL Code is data too

COBOL code can be **queried** for important questions:

- trying to understand how the system works and what it's structure has become
- trying to learn where and how the "what" requirements are implemented
- not off-the-shelf analysis tools, but **questions** asked with **contextual knowledge**

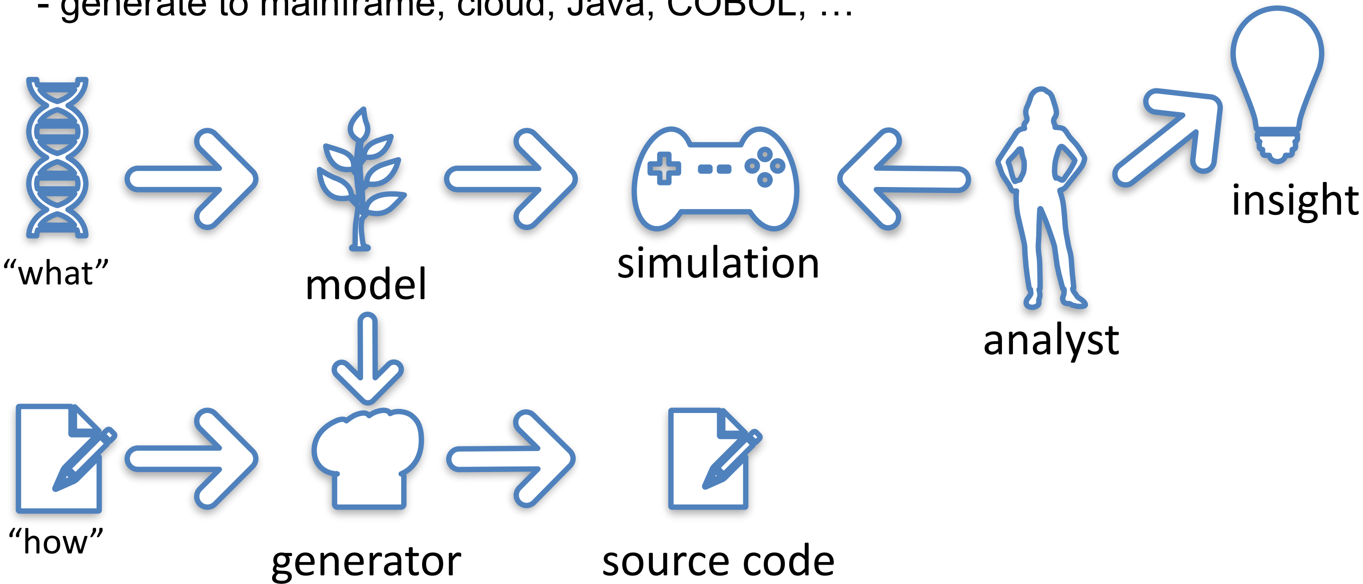


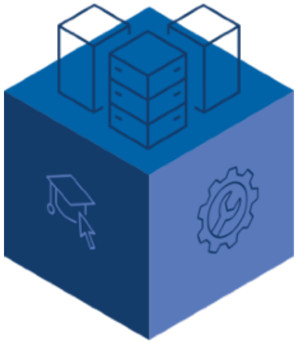


COBOL Code is data too

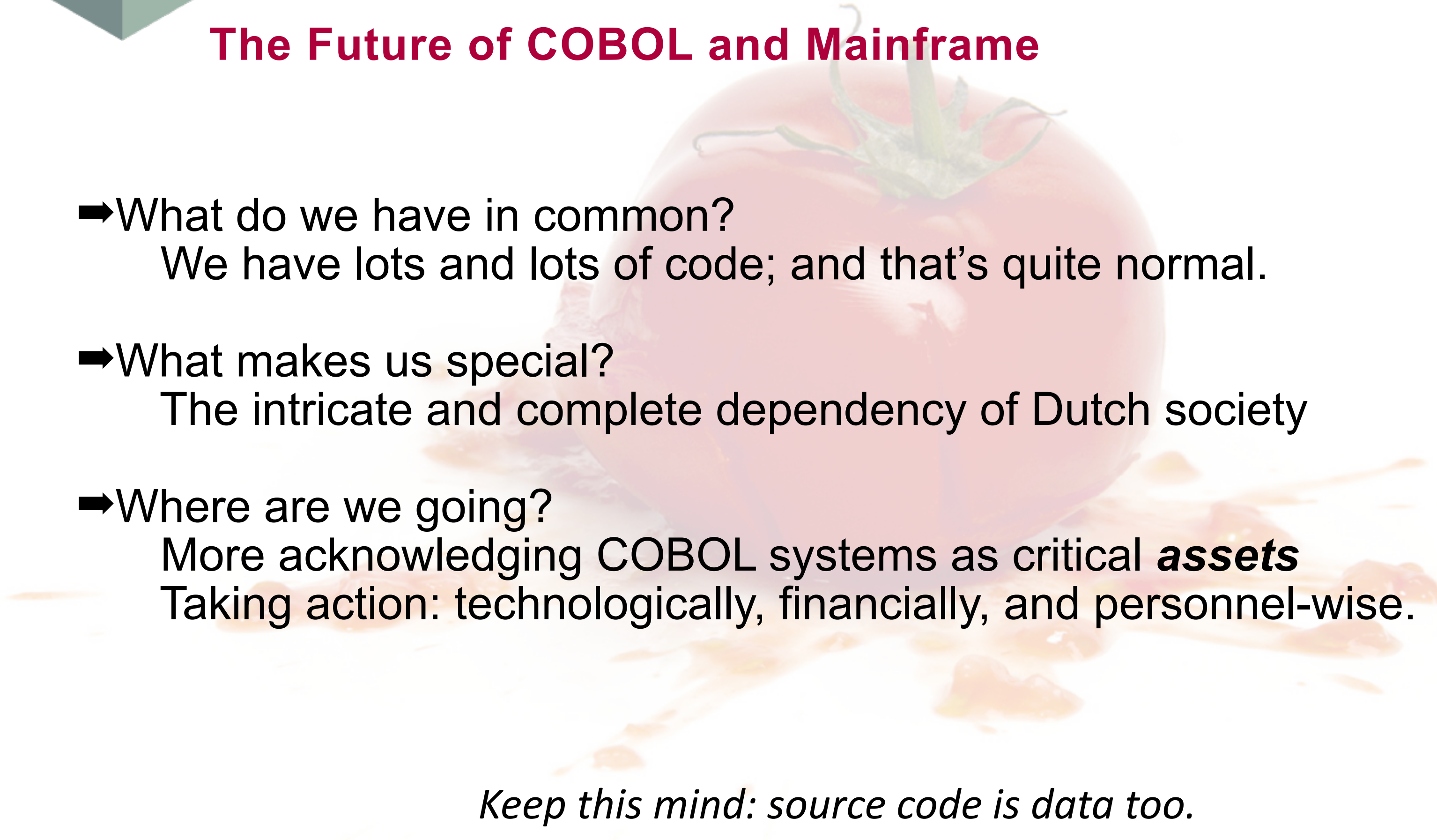
COBOL code can be **simulated** and even **generated**:

- separate the “what” in Domain Specific Languages
- run simulations, visualizations, and predictions, *independent of implementation technology*
- generate to mainframe, cloud, Java, COBOL, …





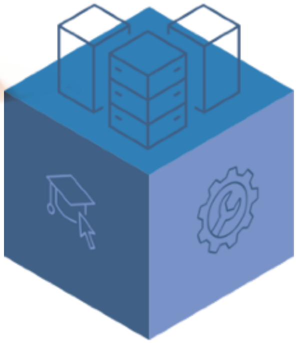
The Future of COBOL and Mainframe



➡What do we have in common?
We have lots and lots of code; and that's quite normal.

➡What makes us special?
The intricate and complete dependency of Dutch society

➡Where are we going?
More acknowledging COBOL systems as critical ***assets***
Taking action: technologically, financially, and personnel-wise.



Keep this mind: source code is data too.