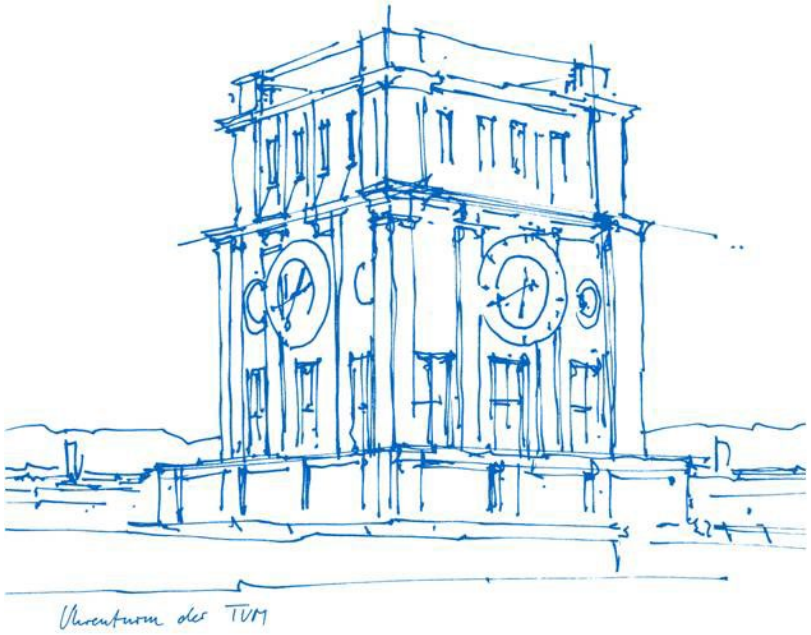
# Will generative AI replace software engineers?

Alexander Pretschner

TUM | fortiss | bidt | CDTM | onetutor.ai

pretschner@tum.de



# Isn‘t this trivial?

BBC

Home News Sport Business Innovation Culture Travel Earth Video Live

## Suffolk expert says AI will replace human coders in 10 years

11 February 2024

By Jon Wright,

Share

Mr Applegate wo

A Suffolk c
human cod

Frankfurter Allgemeine

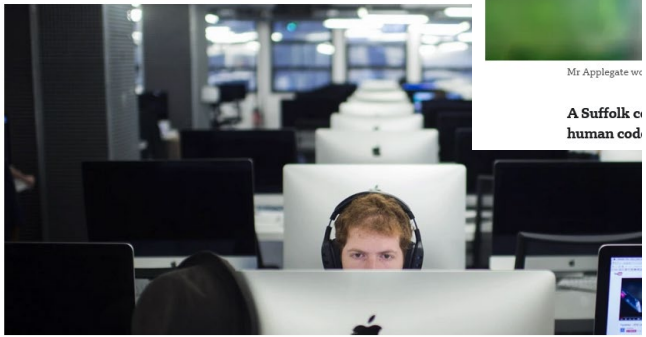
ZEITUNG MEHR F.A.Z.

Wirtschaft › Künstliche Intelligenz › KI und Zukunft des Programmierens: Anforderun

KÜNSTLICHE INTELLIGENZ

## Programmierer haben ausgedient

Von Alexander Wulfers 21.02.2024, 09:39 Lesezeit: 6 Min.



**Lern programmieren und du hast einen sicheren Job, hieß es lange. Plötzlich schreibt die Künstliche Intelligenz den Code. Was nun?**

TECH

## Mark Zuckerberg says AI could soon do the work of Meta's midlevel engineers

Lakshmi Varanasi Jan 11, 2025, 7:28 PM MEZ

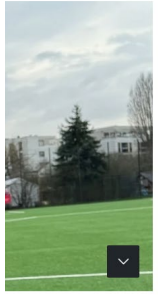
Share Save

**Meta CEO Mark Zuckerberg says the work of midlevel software engineers can soon be outsourced to AI.** BRENDAN SMIALOWSKI/Getty Images

Abo

iisse

–



Tausende

# Today‘s story

I. Software Engineers

II. Software Engineering

III. Where (I don‘t think) genAI can take over

GenAI will assist, not replace. You read this everywhere – everyday. **I‘ll try to understand why.**

Main argument upfront:
SW Engineers make explicit and conscious choices of *what we intend*. Not understanding our intentions and off-loading these choices to an AI likely leads to incorrect or inadequate results …
… which explains varying evidence on productivity gains and which conflicts with (my understanding of) digital humanism in that we *refuse agency and dodge responsibility*.

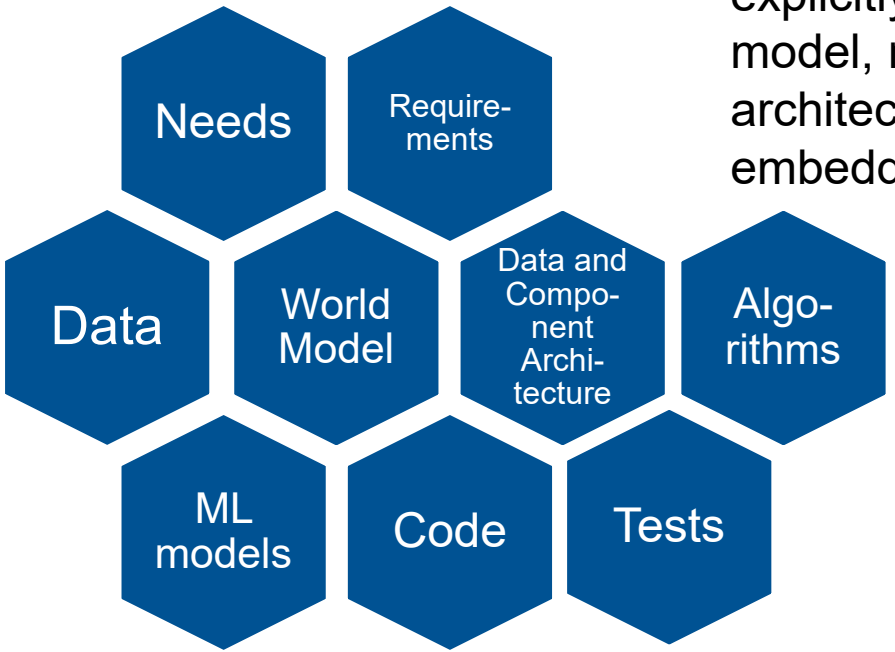
# Today‘s story

**I. Software Engineers**

II. Software Engineering

III. Where (I don‘t think) genAI can take over

# Software-Intensive Systems

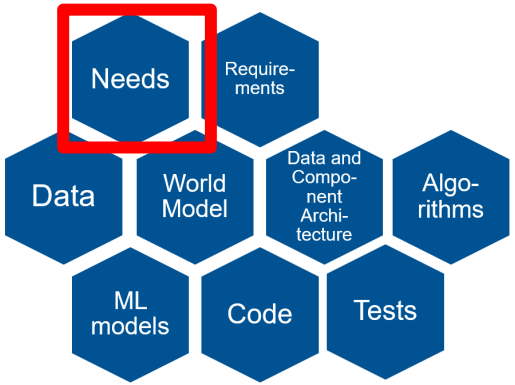



Artifacts need not exist explicitly – conceptual world model, resulting data model, architecture may be directly embedded in the code

# Software Engineers?

Depending on the context,

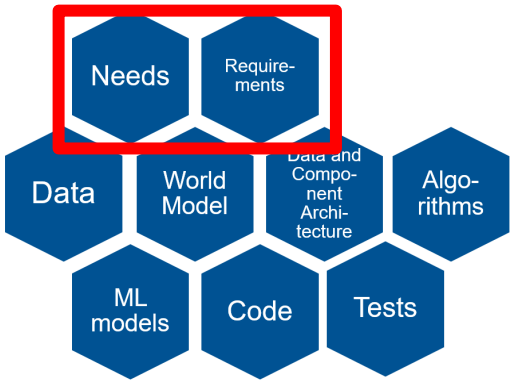
- **elicitation of *needs* may be done by product managers … or requirements engineers or coders**



# Software Engineers?

Depending on the context,

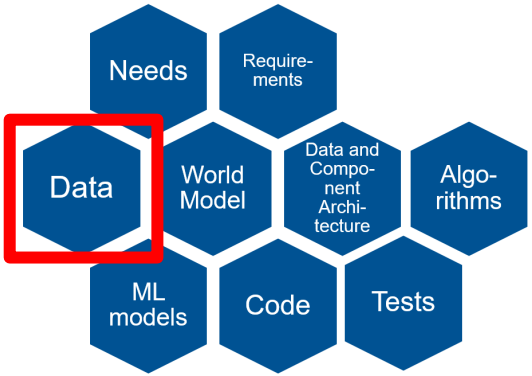
- elicitation of needs may be done by product managers … or requirements engineers or coders
- **requirements engineers may turn needs into *requirements* - that come as vague user stories or user req specs or system req specs**



# Software Engineers?

Depending on the context,

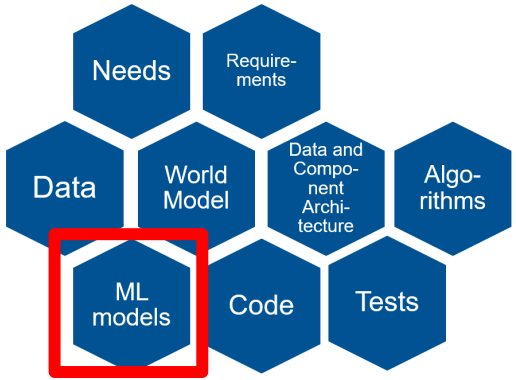
- elicitation of needs may be done by product managers … or requirements engineers or coders
- requirements engineers may turn needs into requirements - that come as vague user stories or user req specs or system req specs
- **data analysts may cleanse and prepare *data***



# Software Engineers?

Depending on the context,

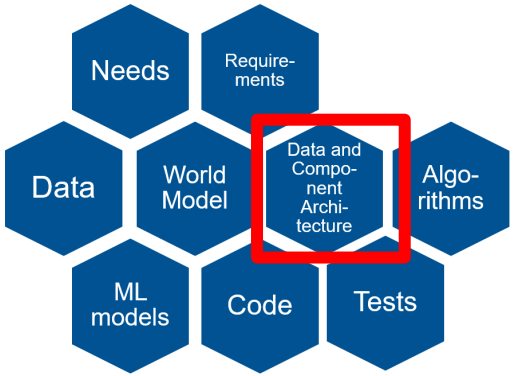
- elicitation of needs may be done by product managers … or requirements engineers or coders
- requirements engineers may turn needs into requirements - that come as vague user stories or user req specs or system req specs
- data analysts may cleanse and prepare data
- **AI experts may train and validate *ML models***



# Software Engineers?

Depending on the context,

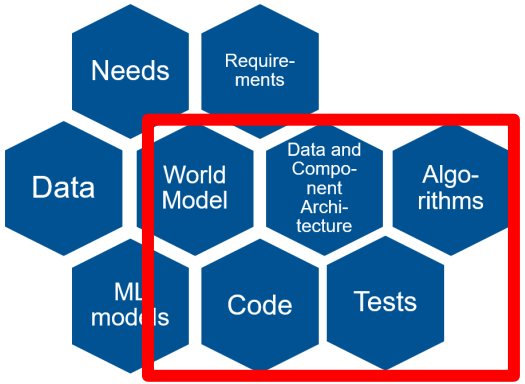
- elicitation of needs may be done by product managers … or requirements engineers or coders
- requirements engineers may turn needs into requirements - that come as vague user stories or user req specs or system req specs
- data analysts may cleanse and prepare data
- AI experts may train and validate ML models
- **architects build world models, create *data models* and design *architectures* – that come as explicit artifacts or as code**



# Software Engineers!

Depending on the context,

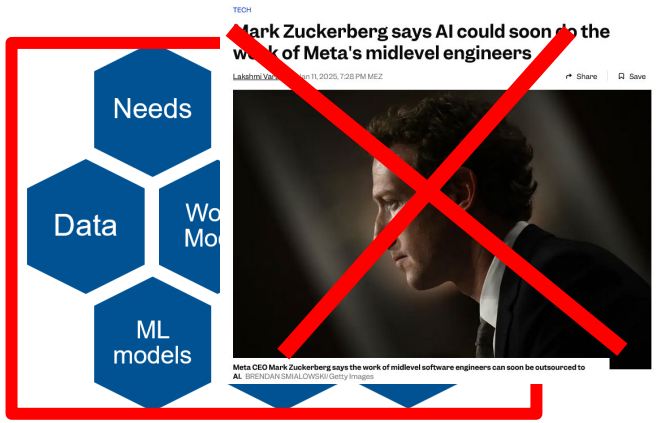
- elicitation of needs may be done by product managers … or requirements engineers or coders
- requirements engineers may turn needs into requirements - that come as vague user stories or user req specs or system req specs
- data analysts may cleanse and prepare data
- AI may experts train and validate ML models
- architects build world models, create data models and design architectures – that come as explicit artifacts or as code
- **developers write *code* and *tests* …**



# Software Engineers!

Depending on the context,

- elicitation of needs may be done by product managers … or requirements engineers or coders
- requirements engineers may turn needs into requirements - that come as vague user stories or user req specs or system req specs
- data analysts may cleanse and prepare data
- AI may experts train and validate ML models
- architects build world models, create data models and architectures – that come as explicit artifacts or as code
- **developers write code and tests**
  **… but may do all of the above!**



# Flavors of Software Engineers

*Total laymen* who didn‘t code before („soccer dads“): nobody to be replaced

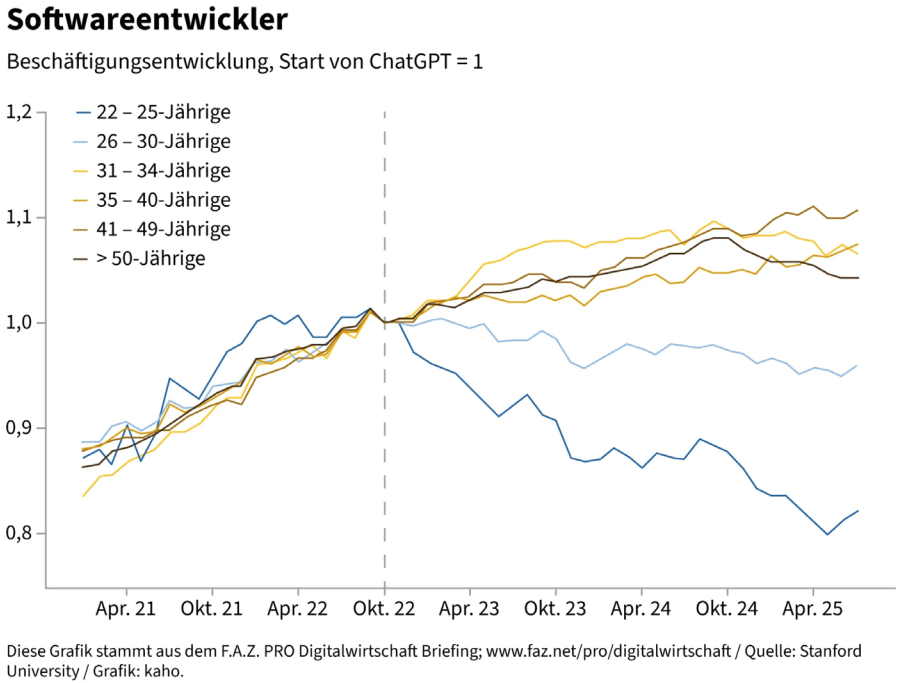
*Knowledgeable but not trained* („music teacher turned website developer“):
write simple programs, may indeed be replaced

*Professionals but not trained in software* („electrical engineers doing software“):
know what they are doing, understand business domain, probably will be assisted but not replaced

*Professionals with serious SW engineering education*: will be assisted but not replaced

These come at *different levels of seniority*

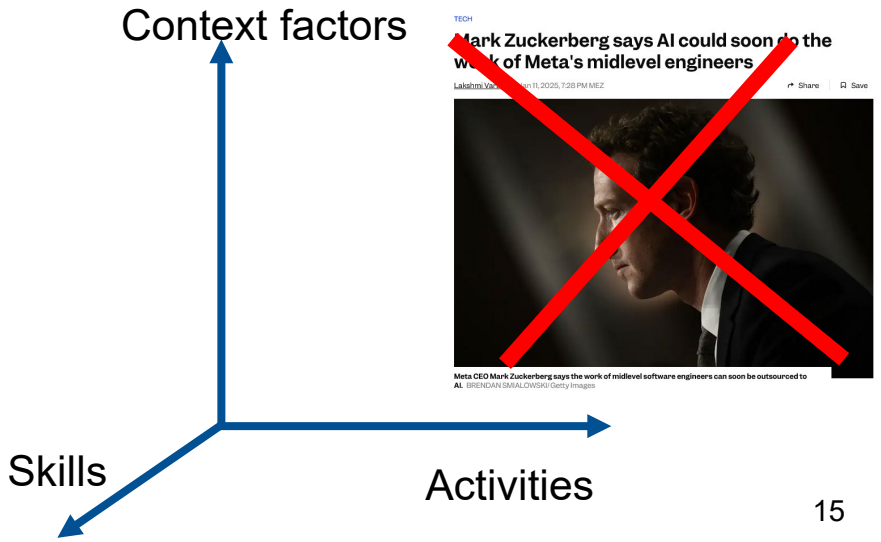
# How Employment Develops




https://digitaleconomy.stanford.edu/publications/canaries-in-the-coal-mine/

# So?

- Software engineers perform different activities, creating various implicit or explicit artifacts
- Software engineers range from laymen to professionals with many shades
- Software engineers come with different skill levels and at different levels of seniority
- Software engineers work in a highly context-specific manner:
  - **Application domain**
    - Risk
      - Safety or security criticality
      - Ease of update
      - Regulation
      - „Private“ code vs. deployed products
    - Level of „standardness“
    - Trade-Offs
  - **Development process**
  - **(Legacy) tech stack**



# Today‘s story

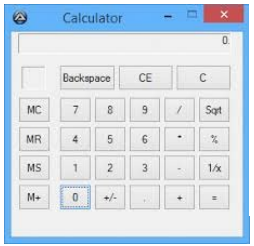
I. Software Engineers

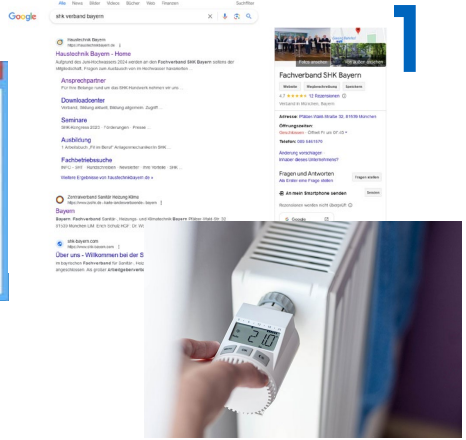
**II. Software Engineering**

III. Where (I don‘t think) genAI can take over

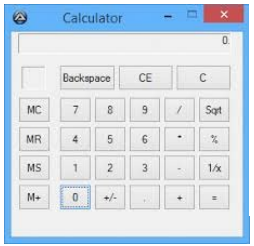
# What is Software?

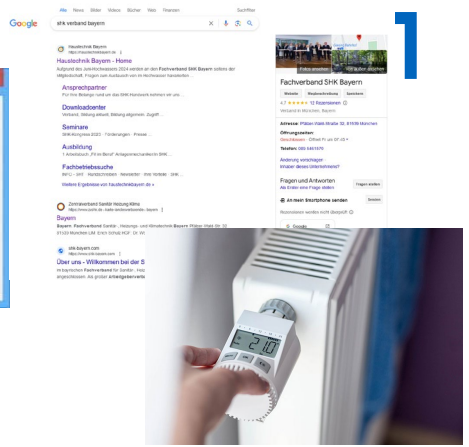
Software implements „functions.“ Functions map input to output.



# What is Software?




Software implements „functions.“ Functions map input to output.
What a function is supposed to compute is described in a „specification.“
„Implementation“ is step-by-step by an *algorithm* or by a *machine-learned model*.

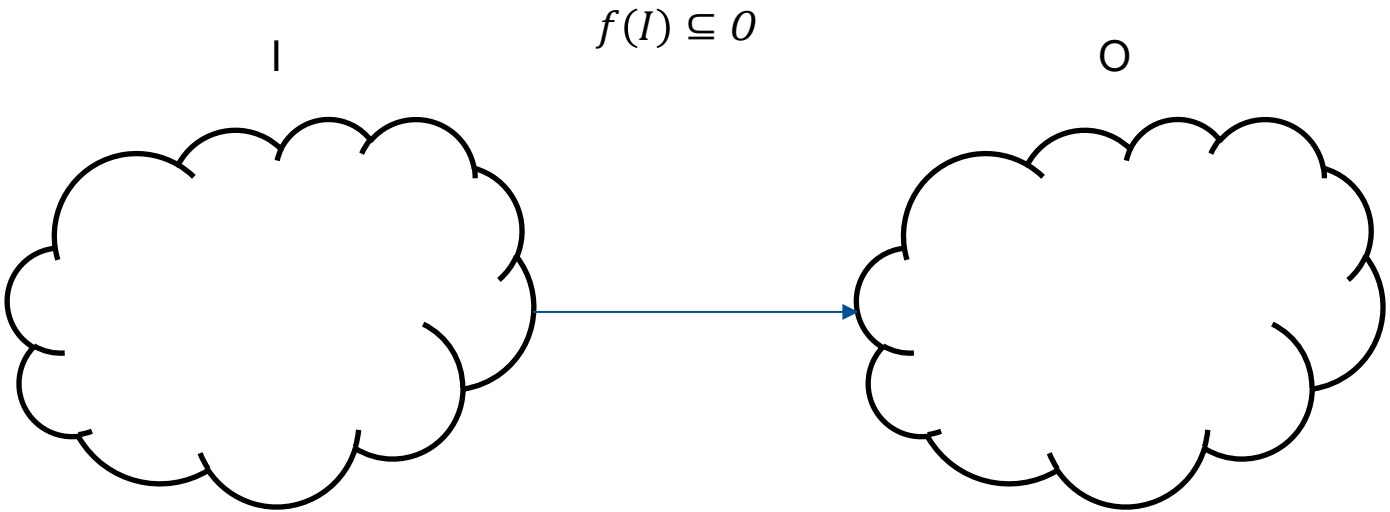
Crucial design principle: divide-et-impera. Break down a big problem into smaller problems. Repeat.

For every function, there are many different implementations. These implementations differ

- Structurally in how the original big problem was recursively broken down („refined“);
- W.r.t. the chosen algorithm or machine-learned model;
- Therefore w.r.t. resource consumption (time, memory) as different algorithms can implement the same function; or w.r.t. correctness for ML models; and w.r.t. further extra-functional properties: maintainability, security

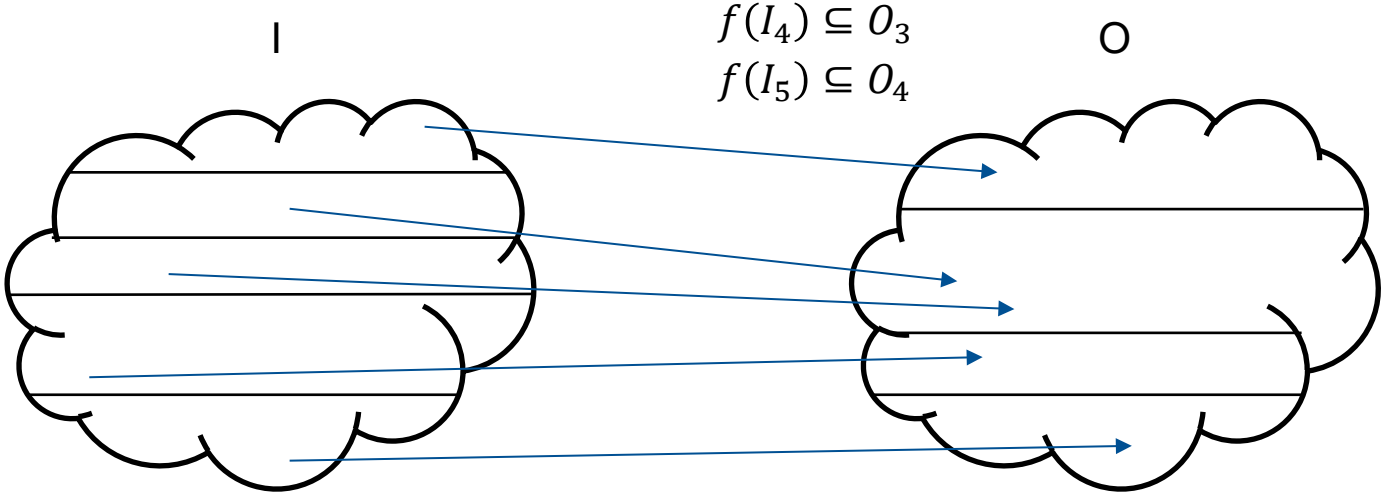
SW engineers take the underlying design decisions – there usually is not the one best decision!
SW engineers explicitly design trade-offs w.r.t. implicit or explicit goals. **Who comes up with these goals?**

# Specifications are Functions



# Specification: Use cases

$$f(I_1) \subseteq O_1$$
$$f(I_2) \subseteq O_2$$
$$f(I_3) \subseteq O_2$$
$$f(I_4) \subseteq O_3$$
$$f(I_5) \subseteq O_4$$




$$I = I_1 \cup \cdots \cup I_n$$

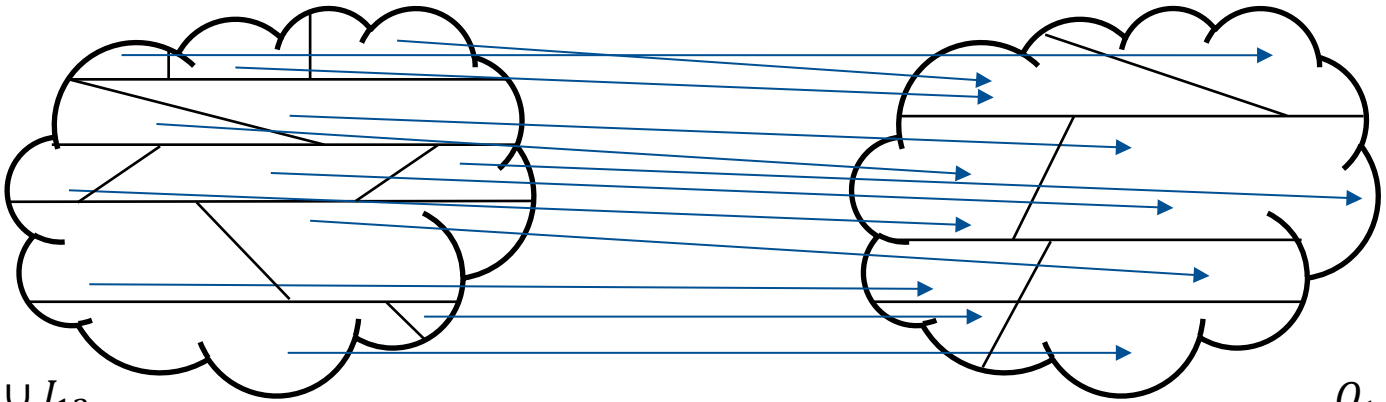
$$O = O_1 \cup \cdots \cup O_m$$

# Specification: Refinement

$f(I_{11}) \subseteq O_{12}$
$f(I_{12}) \subseteq O_{11}$
$f(I_{13}) \subseteq O_{11}$
...

I

O

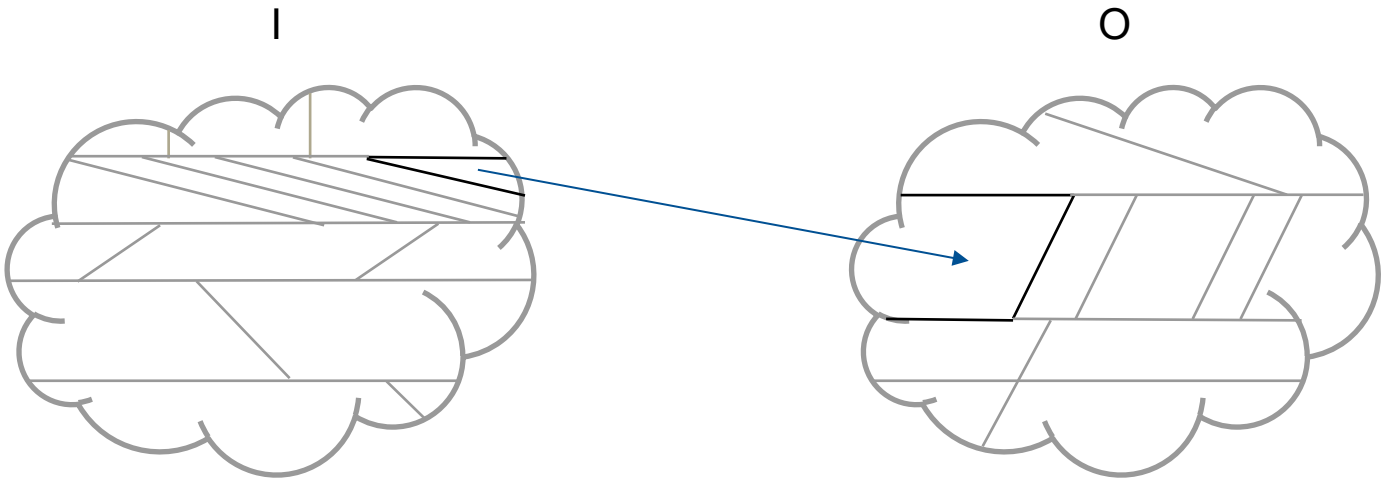


$I_1 = I_{11} \cup I_{12} \cup I_{13}$
$I_2 = I_{21} \cup I_{22}$
$I_3 = I_{31} \cup I_{32} \cup I_{33}$
...

$O_1 = O_{11} \cup O_{12}$
$O_2 = O_{21} \cup O_{22}$
...

# Specification: Refinement

$$f(I_{226}) \subseteq O_{221}$$



# Approaches

Top-down [70s]

Linear, mathematically founded „stepwise refinement“: correct-by-design

Bottom-up [90s]

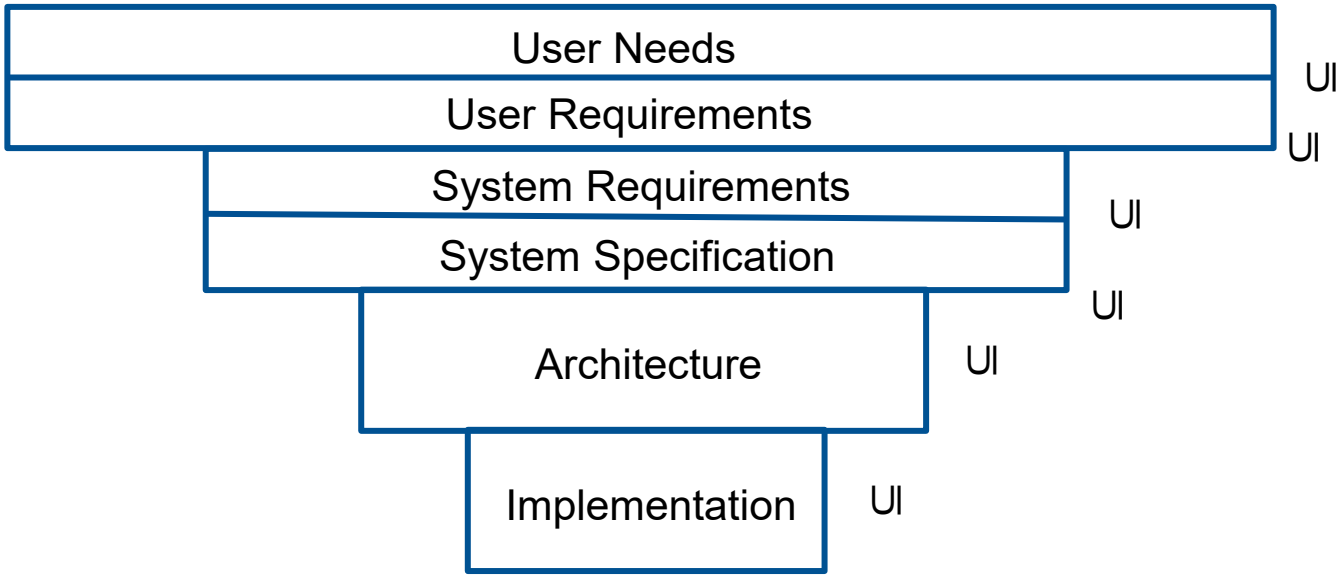
Non-linear agile software development

In-between

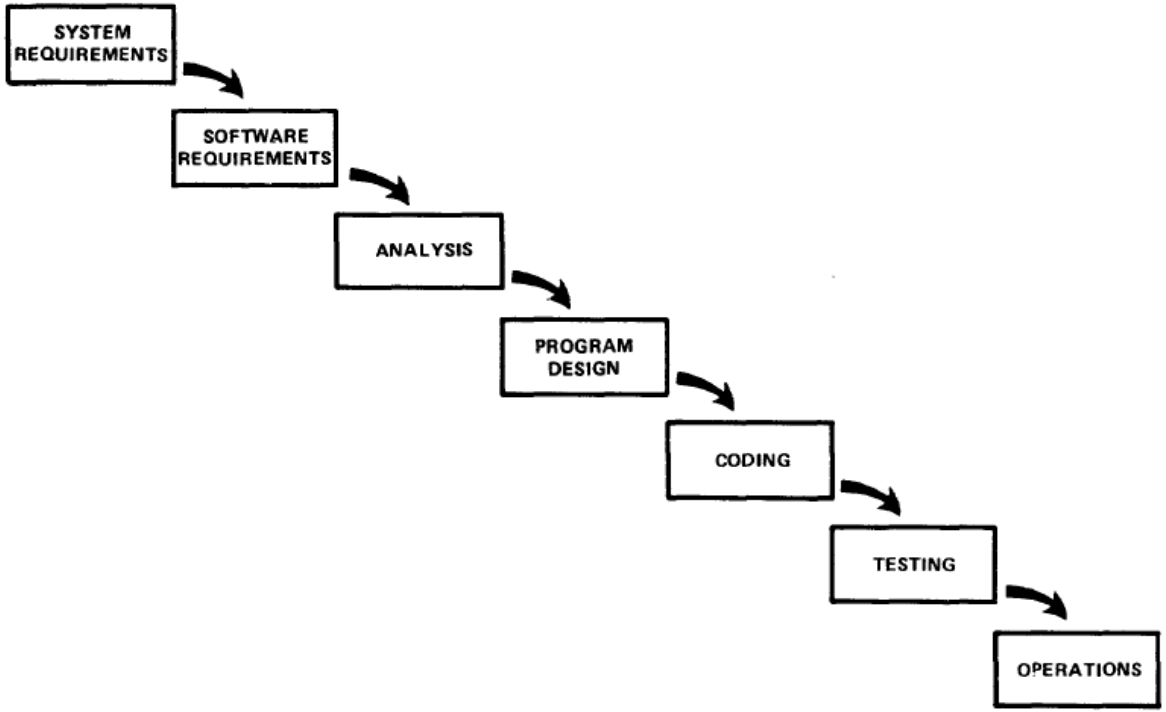
„Stepwise refinement“ à la Wirth: divide-et-impera – and get back where necessary

“Each refinement implies a number of design decisions based upon a set of design criteria. Among these criteria are efficiency, storage economy, clarity, and regularity of structure. Students must be taught to be conscious of the involved decisions and to critically examine and to reject solutions, sometimes even if they are correct as far as the result is concerned; they must learn to weigh the various aspects of design alternatives in the light of these criteria**. In particular, they must be taught to revoke earlier decisions, and to back up, if neccessary even to the top.** Relatively short sample problems will often suffice to illustrate this important point; it is not necessary to construct an operating system for this purpose.” [Wirth: Program Development by Stepwise Refinement, CACM **1971**]

# Refinement of Software Artifacts

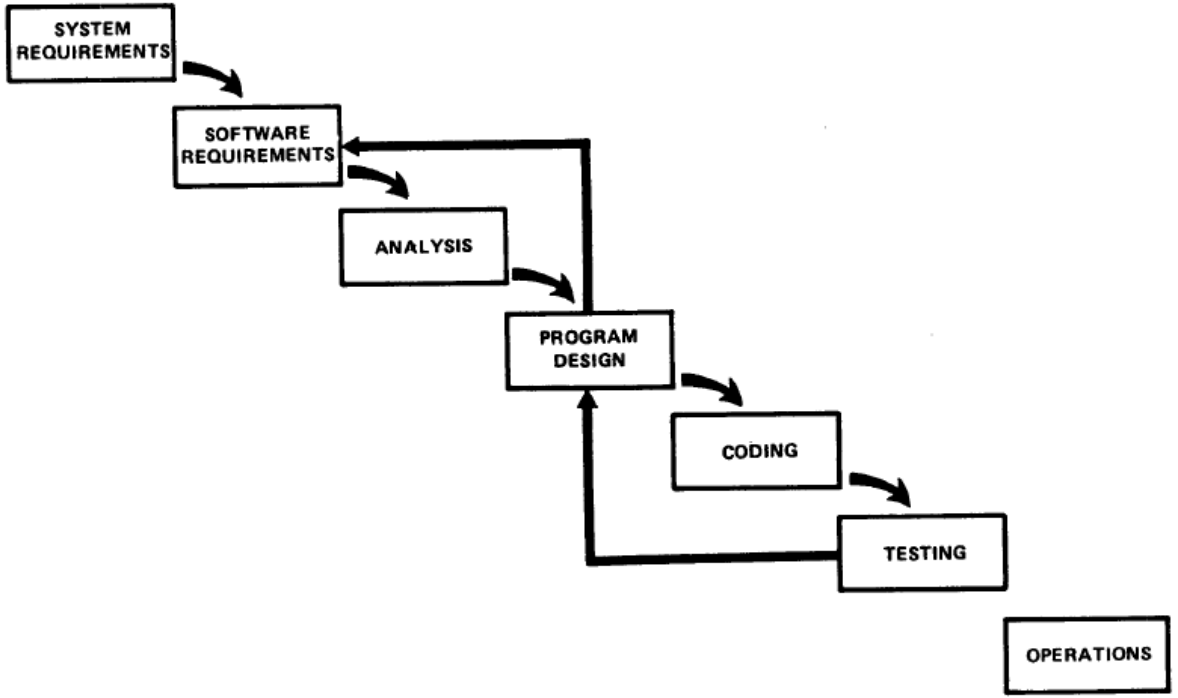


# … the Waterfall




W. Royce: Managing the development of large software systems
Proc. IEEE WESCON, pp. 1-9, 1970

# No Linear Process: Change and Error




W. Royce: Managing the development of large software systems

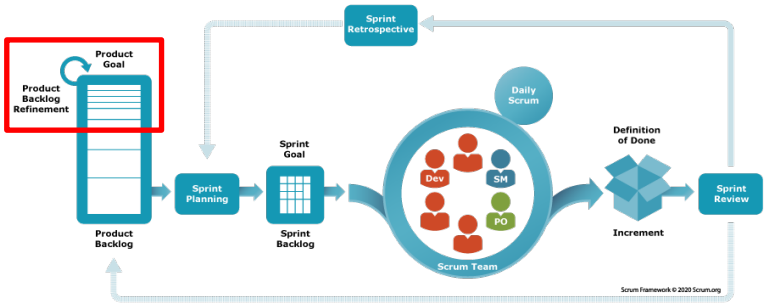
Proc. IEEE WESCON, pp. 1-9, **1970**

# Agility

Linear refinement assumes a more or less complete set of requirements upfront and then **correct refinement decisions** in each step

In reality: requirements often not known up-front and changing

**Agility adresses lack of knowledge** and therefore lack of linearity and plannability via ongoing customer/user interactions




**„Requirements specification“ iteratively co-evolves with code, and necessarily does so!**

# Understanding over Time


# … and the Point?

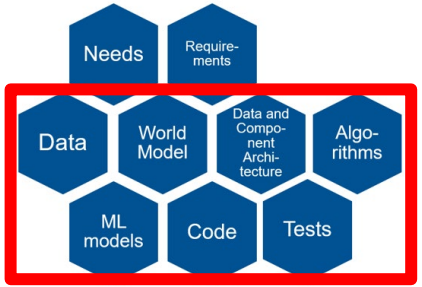
For non-trivial and not entirely standard systems, it is not the case that we write down a specification or a prompt upfront, then start coding or push a button, and get the system implementation!

Instead, as we iteratively develop the system, we understand what we want to build – in terms of functionality, in terms of structure, in terms of extra-functional properties.

And we decide on design alternatives.

Which we can only do if we understand the business domain –
and software engineering: the impact of different design decisions

*This can be assisted but not replaced by genAI.*



# Understanding and Designing

SW engineering is about **taking decisions and trade-offs**

**Each refinement implies a number of design decisions based upon a set of design criteria. Among these criteria are efficiency, storage economy, clarity, and regularity of structure. Students must be taught to be conscious of the involved decisions and to critically examine and to reject solutions**, sometimes even if they are correct as far as the result is concerned; they must learn to weigh the various aspects of design alternatives in the light of these criteria. In particular, they must be taught to revoke earlier decisions, and to back up, if necessary even to the top. Relatively short sample problems will often suffice to illustrate this important point; it is not necessary to construct an operating system for this purpose. [Wirth 1971]

# Understanding and Designing

SW engineering is about **taking decisions and trade-offs**

**Each refinement implies a number of design decisions based upon a set of design criteria. Among these criteria are efficiency, storage economy, clarity, and regularity of structure. Students must be taught to be conscious of the involved decisions and to critically examine and to reject solutions**, sometimes even if they are correct as far as the result is concerned; they must learn to weigh the various aspects of design alternatives in the light of these criteria. In particular, they must be taught to revoke earlier decisions, and to back up, if necessary even to the top. Relatively short sample problems will often suffice to illustrate this important point; it is not necessary to construct an operating system for this purpose. [Wirth 1971]

Who takes this decision? Possibly an LLM.

But w.r.t. which goal? And who understands and specifies this goal?

# Technical and Business Domains

Software Engineers need to understand technology and business domains

Technological abstractions captured in frameworks/languages/patterns
… and in the developers‘ heads
Works well.


Business domain captured in the developers‘ heads
… and in DSLs/rule sets/analysis patterns
Didn‘t work so well pre-LLM. How would it? Too much contextual variation.

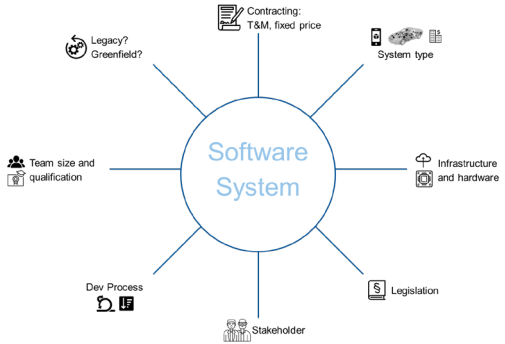
# Today‘s story

I. Software Engineers

II. Software Engineering

**III. Where (I don‘t think) genAI can take over**

# AI-Generated Code



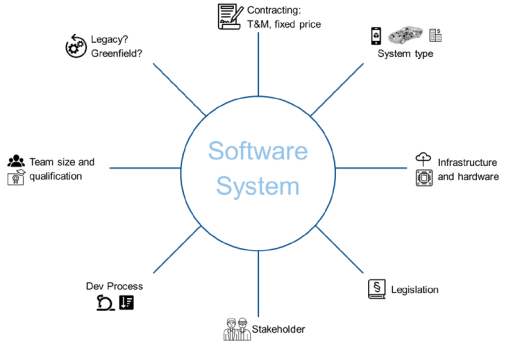

Describe functionality and structural preferences as a prompt.

Add further context to prompt – for both technology and business domains, in terms of functionality, extra-functional properties, trade-offs.

Then either generate code or intermediate format.
Verify result.
Iterate.

More context usually improves AI-generated code.

# AI-Generated Code

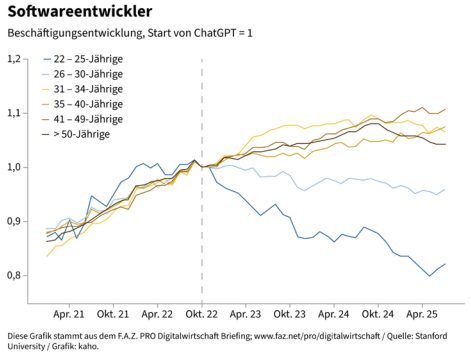



Describe functionality and structural preferences as a prompt.

Add further context to prompt – for both technology and business domains, in terms of functionality, extra-functional properties, trade-offs.

Then either generate code or intermediate format. Iterate. More context usually improves AI-generated code.

*But an AI cannot know what we want.*
It can only pick a statistically likely (== most standard) solution.
Junior or non-professional SW engineers do the same.
In contrast, *senior and professional SW engineers understand what is non-standard in their specific context!*

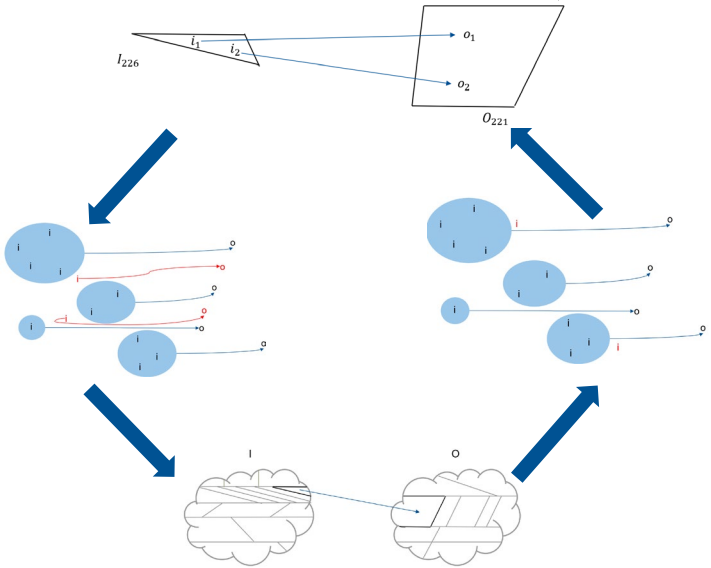


# Hallucinations, Trust, Verification

creation

verification

# Think as you Specify – Think as you Code  - Verify as you Think



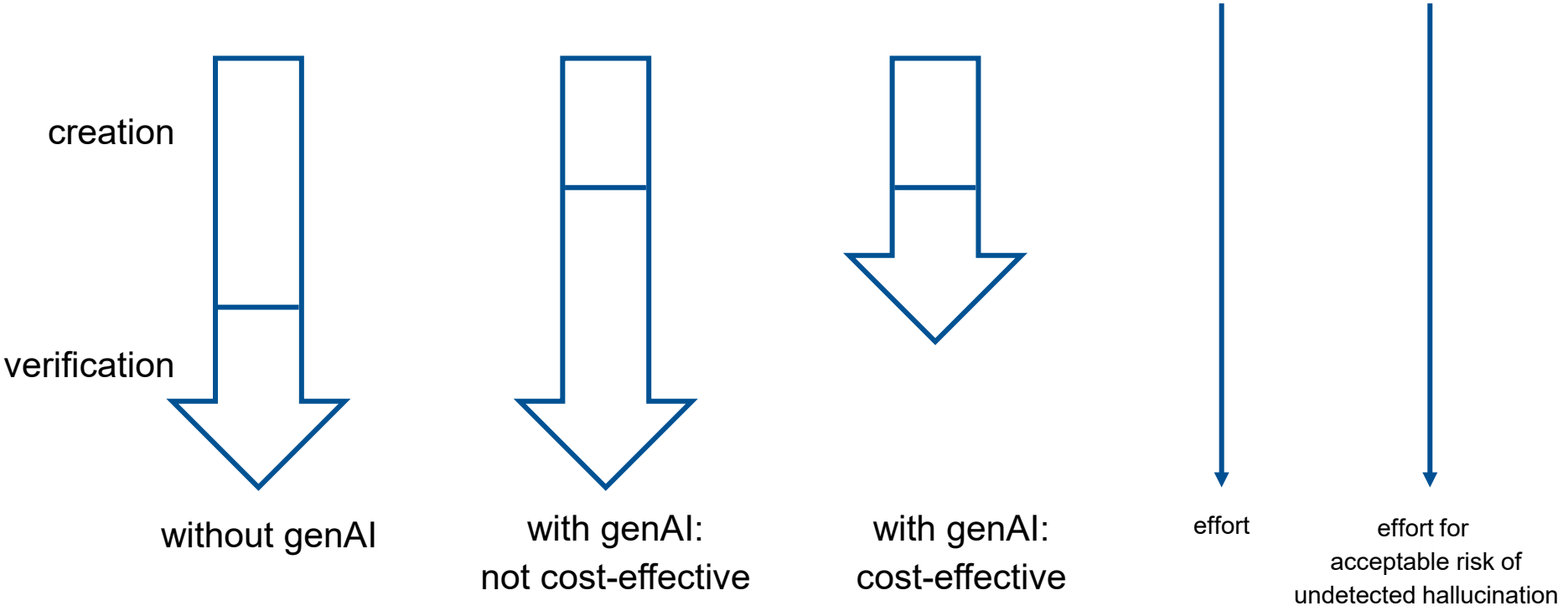


Co-creation: Start with rough specification – get partly inadequate results – verify result and fill in/correct information – iterate

# Think as you Specify

Under which circumstances is it faster/better/more convenient to iterate on a specification and check the generated code than to directly iterate on the code?

# Cost Effectiveness



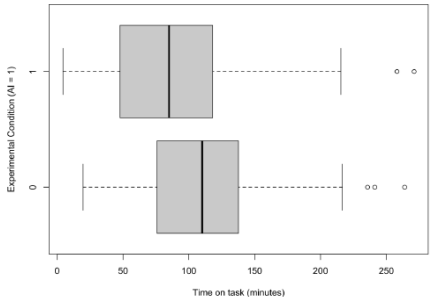
# Cost Effectiveness, Productivity Gains

Today‘s answer: Depends!

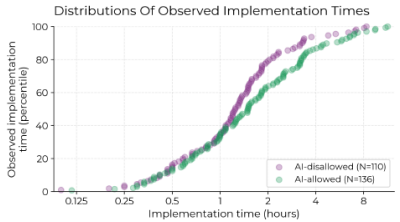
Works well for standard problems (specifically, „boilerplate code“);
helps with exact usage of libraries; creates simple scripts; etc.

Often works well for coming up with a first idea for a solution;
or suggesting a re-structuring of the code.
*Requires interaction* though!

Then, grossly differing statements about productivity gains.
Likely reason: vastly differing kinds of software.




https://arxiv.org/pdf/2410.12944
Speedup of 21% observed




https://arxiv.org/abs/2507.09089
Slowdown of 19% observed

# Beyond Economics

The more standard a problem, the better AI-generated results will be, at least in terms of functional correctness. Simple low-risk problems can be off-loaded to the machine. The machine may also help with refining a big problem into small problems.

But software engineering is about

- understanding what is to be implemented … which needs to be done in an iterative way; and about
- taking decisions about trade-offs (e.g., „security vs. performance“)

How would an AI „decide“? *Is it ethical to not know which decisions were taken and why?*
This conflicts with (my understanding of) digital humanism in that we then *refuse agency and dodge responsibility*. *Machines cannot assume responsibility.*

(But do all human software engineers know what they are doing? Different debate on responsibility …)

# In Sum: Why we need Humans

Software development: iterative process of „refinements“ with learning and possible corrections

Many possible refinements at any stage of development

- Differ w.r.t. various properties
- Goals determine whether one refinement is „correct,“ or more „adequate“ than another
- Need to take decisions. Difficult upfront: problem necessarily not sufficiently well understood yet

If correctness impossible to determine, or goals not explicit, or trade-offs unclear,
why pick one rather than another?
→ AI (and a junior or unaware developer) will blindly pick any: a statistically likely refinement

Does not matter sometimes. Does matter sometimes.