



# Rewriting the Role Developers in the Age of LLMs

Horacio Gonzalez

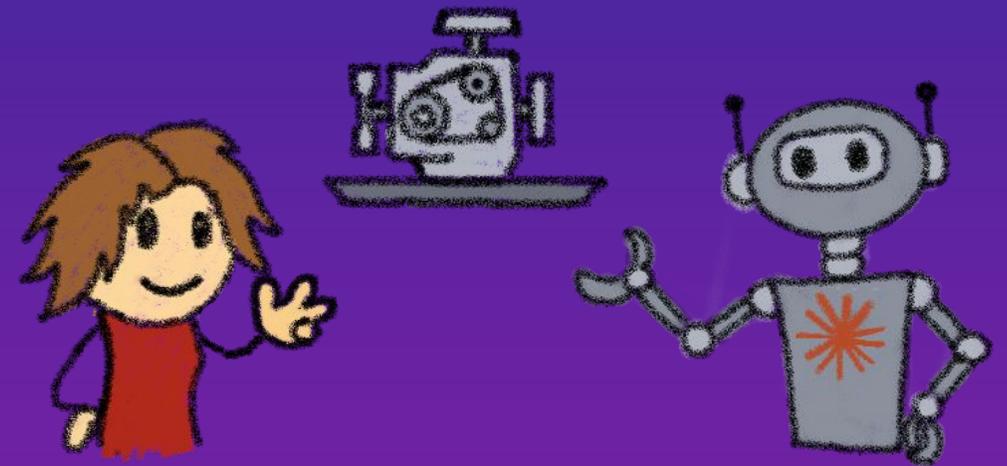
2026-02-10



@LostInBrittany

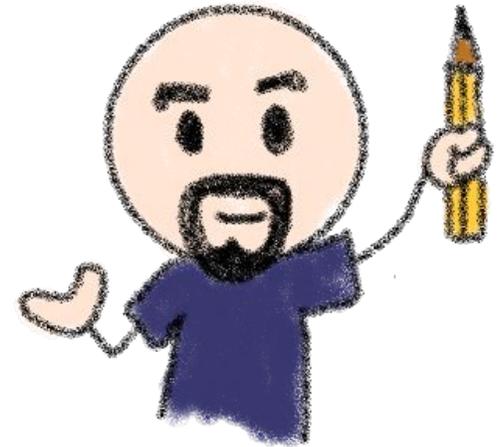
# Rewriting the Role

Developers in the Age of LLMs



# What are we going to talk about?

- Programmers Are Always Doomed...  
Until they are not
- When Tools Learn, So Must We  
Deskilling or reskilling in the age of AI
- The Developer's New Workflow  
Co-authoring with the machine
- The Developer's Journey  
Growing up with smarter tools
- Teaching the Next Generation  
How do we teach programming when the computer can already code?
- Differently Human  
The future of software development



# This talk is dedicated to my father



If I work in IT today, it's only because I followed in his footsteps.

# Programmers Are Always Doomed...

## Until They're Not



# The first program language\*: Fortran

It will make make programmers obsolete!



\* Grace Murray Hopper invented the concept and tools that made high-level programming possible, Fortran was the first full implementation of that idea

# The first program language\*: Fortran



From machine code:

```
0001 0000 0000 10101    ; LOAD constant 21 into register A
0001 0001 0000 00010    ; LOAD constant 2 into register B
0010 0000 0001 00010    ; MULTIPLY A × B → result in register A
0011 0000 0000 10000    ; STORE result (42) into memory address 16
```

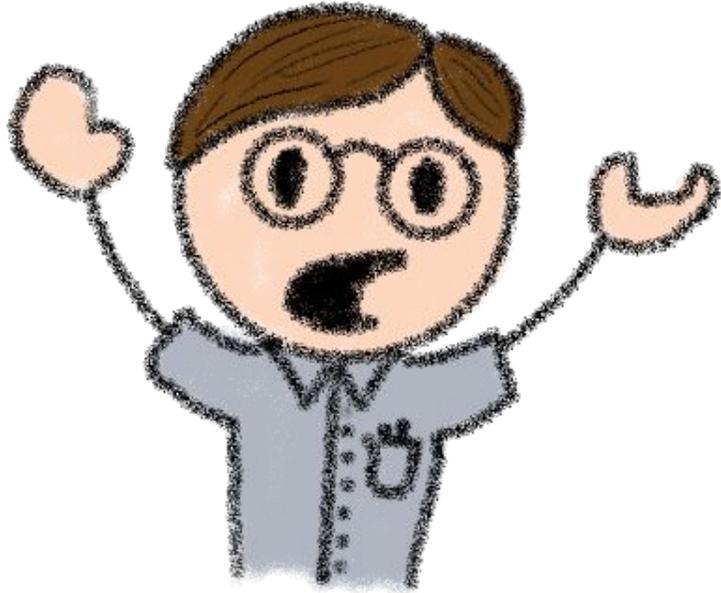
Low-level: registers, opcodes, memory addresses

To Fortran:

```
INTEGER A, B, C
A = 21
B = 2
C = A * B
PRINT *, C
END
```

Low-level: registers, opcodes, memory addresses

# The first program language: Fortran



Fortran is not real programming!

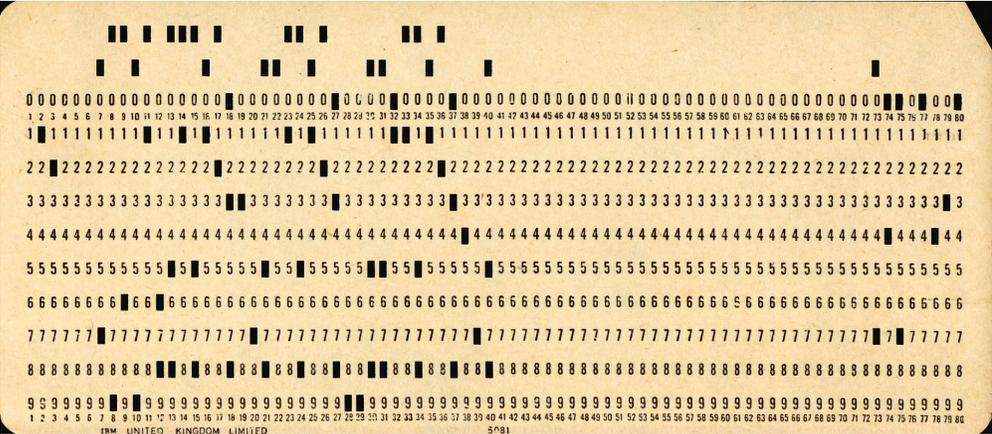
Anybody can write `DO 10 I=1,10`  
without understanding anything about  
opcodes, registries or memory.

Programming is dead!

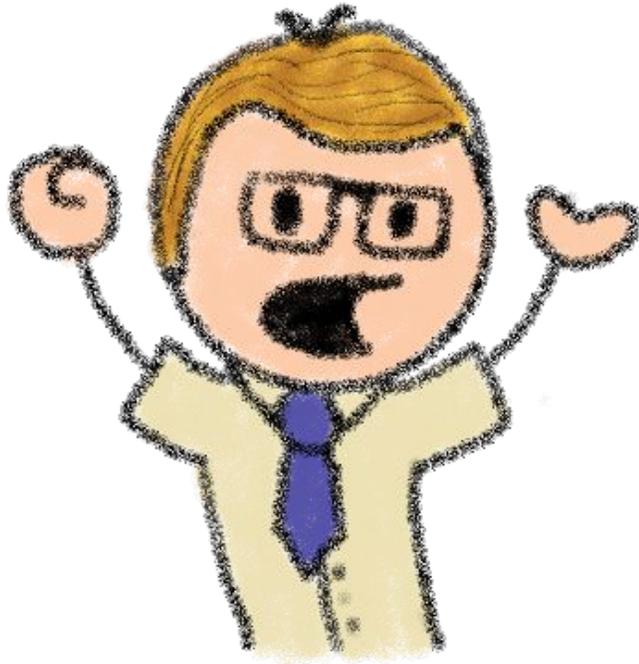
Source: IBM history of Fortran  
<https://www.ibm.com/history/fortran>

# From punched cards to keyboards

Programmers won't think before coding anymore!



# From punched cards to keyboards



I am not a typist !  
Code is meant to be hard,  
an intellectual pursuit,  
typing it is for clerks.  
Programming is dead !



# From punched cards to keyboards



**SIMPLAFIND** — a motorized, automatic card file—uses existing records without recopying or alteration. For tabulating and other standard or special size cards, sheets, address plates, etc.

**PUSH-BUTTON CONTROL** — at a single touch — the cards you want are quickly and automatically positioned before you. They always come via the shortest route — and fast — less than 3 seconds average time. SIMPLAFIND is unbelievably compact. SIMPLAFIND capacity depends on the card size housed and ranges from 375 to over 1900 linear filing inches per machine, with 553 linear filing inches for tabulating size cards filed horizontally.

**SO EASY** — everything comes to you effortlessly. Removable angle — ended card trays with compressor-followers are only 10" long—light weight—easy to reach—easy to read.

**NO OTHER FILE CAN PROVE SUCH CLAIMS**

**Simplafind**  
MOTORIZED 3-SECOND CARD FINDING

ASK FOR PROOF OR SEE A CONVINCING DEMONSTRATION SIMPLA RESEARCH AND MANUFACTURING CO., Inc. 427 Fourth Avenue, Cor. 29th Street, New York 16, N. Y. • LEXington 2-9250



for greater savings...  
you can **RENT** or **BUY** these machines

Many firms have realized substantial savings by using the amazing speed and accuracy of punched-card accounting machines in getting facts that are timely and complete. Now Remington Rand—and only Remington Rand—offers you even greater savings by giving you your choice of three ways to acquire these machines: Outright Purchase, Use-Purchase, or Rental. Under our unique Use-Purchase plan, for instance, your monthly payments are actually less than the corresponding rental fee—yet you own the machines outright after 100 months!

Punched-card accounting offers you great savings over manual methods. Since Remington Rand offers you three plans for obtaining these savings, it's clear that our representative can be truly impartial in pointing out for your consideration the plan best suited to your needs. Call our local office today for free copies of "Purchase vs. Rental" (TM-739) and "Use-Purchase Plan" (TM-752) showing comparative costs. Or, write Management Controls Reference Library, Room 1190, 315 Fourth Avenue, New York 10, N.Y.

**Remington Rand**

## Sources:

- Wikipedia history of Punched Cards

[https://en.wikipedia.org/wiki/Punched\\_card](https://en.wikipedia.org/wiki/Punched_card)

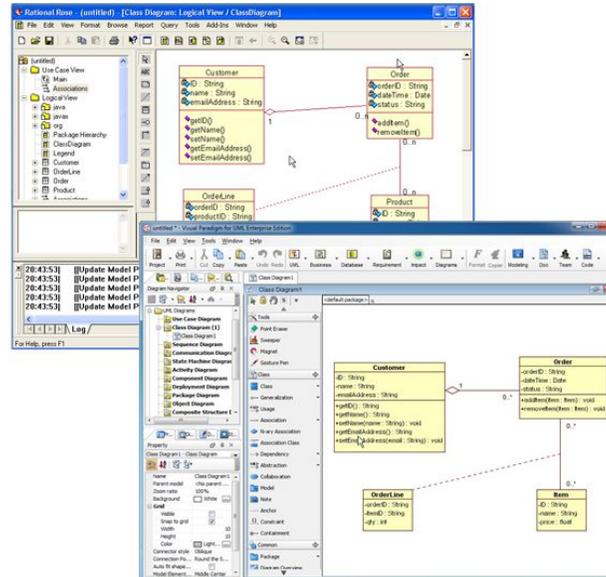
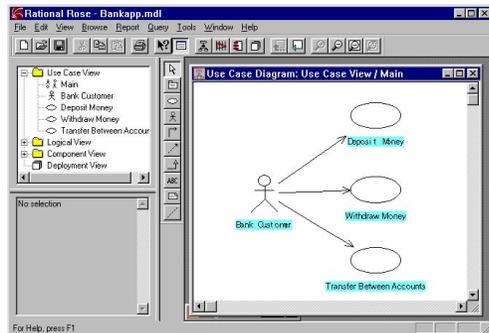
- Punching cards was a clerical job

<https://www.computerhistory.org/revolution/punched-cards/2/4>



# From Code to Models: the Automation Dream

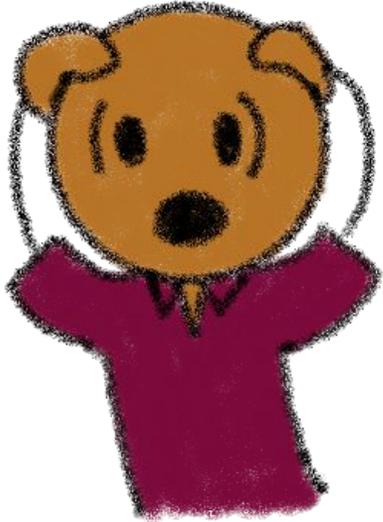
Business users will be able to build applications without programming



MDE  
UML  
CASE  
Rational Rose  
Eclipse EMF



# From Code to Models: the Automation Dream



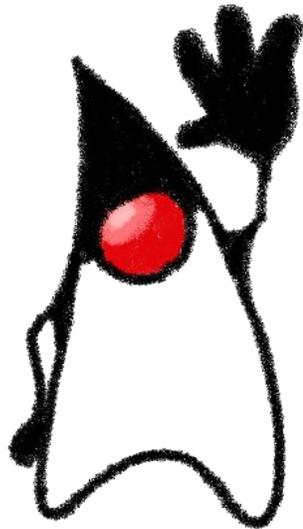
With CASE analysts will just design  
and computers will generate the code!  
Anybody will be able to use visual tools  
to create apps without a single line of code!  
Programming is dead!

Some sources:

- Software Engineering in the Twenty-First Century (M. R. Lowry, 1992)  
<https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1012/930>
- The Last One  
[https://en.wikipedia.org/wiki/The\\_Last\\_One\\_%28software%29](https://en.wikipedia.org/wiki/The_Last_One_%28software%29)

# Write once, run anywhere: Java

Anyone can be a programmer now!



Memory safe  
WORA  
Portability  
Managed runtime  
Applets  
Democratize software development



# Write once, run anywhere: Java



A language so dumb that  
demands no superior intelligence!  
Garbage collector, memory safe,  
huge standard library, clean syntax...  
Real developers will be replaced!  
Programming is dead!

Some sources:

<https://www.joelonsoftware.com/2005/12/29/the-perils-of-javaschools-2/>

[https://nedbatchelder.com/blog/200601/joel\\_spolsky\\_is\\_a\\_crotchety\\_old\\_man.html](https://nedbatchelder.com/blog/200601/joel_spolsky_is_a_crotchety_old_man.html)



# Low-Code / No-Code

Now anyone can be a developer



LOW-CODE



NO-CODE

# Low-Code / No-Code



It's the age of citizen developers!

No knowledge is needed, coding skills

are obsolete, even my grampa will

be able to create apps!

Programming is dead!



# Low-Code / No-Code

## Some sources:

- The Rise of the Citizen Developer  
[https://www.researchgate.net/publication/358383894\\_Rise\\_of\\_the\\_Citizen\\_Developer](https://www.researchgate.net/publication/358383894_Rise_of_the_Citizen_Developer)
- Will low-code/no-code platforms replace traditional developers?  
[https://www.productmarketingalliance.com/developer-marketing/will-low-code-no-code-development-platforms-replace-traditional-developers/?utm\\_source=chatgpt.com](https://www.productmarketingalliance.com/developer-marketing/will-low-code-no-code-development-platforms-replace-traditional-developers/?utm_source=chatgpt.com)
- Will the No Code Movement Change Software Development?  
<https://www.quandarycg.com/no-code-movement-software-development-changes/>

# What Every Abstraction Wave Taught Us

Yet another apocalyptic change, it must be Monday...



# What Every Abstraction Wave Taught Us



- Each new layer of abstraction triggers fear of obsolescence.
- Every time, developers adapt... and redefine the craft.
- Automation doesn't eliminate skill; it changes where it lives.
- From bit-twiddling to system-thinking, we keep moving up the stack.

# What Every Abstraction Wave Taught Us

We don't lose craft,  
we move it up a level

# When Tools Learn, So Must We

## Deskilling or Reskilling in the Age of AI

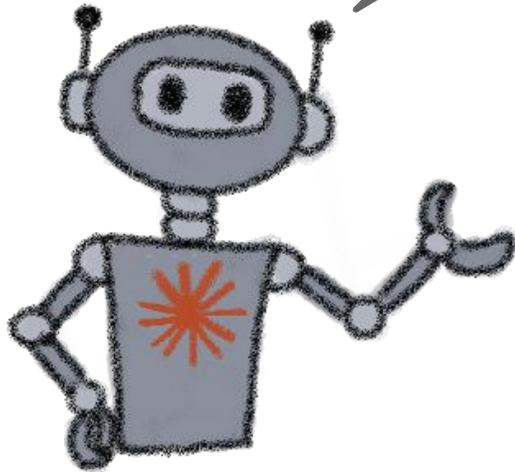


# When Tools Learn, So Must We

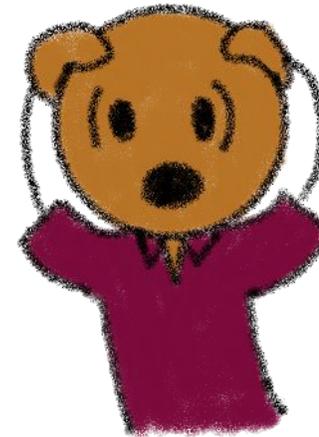


Are we being deskilled – or are we reskilling?

I can code, refactor,  
test, commit, make PRs...



What's left for me?



# The Deskilling Fear

## If AI can code, what's left for developers?



IDEAS

## The Age of De-Skilling

Will AI stretch our minds—or stunt them?

By Kwame Anthony Appiah

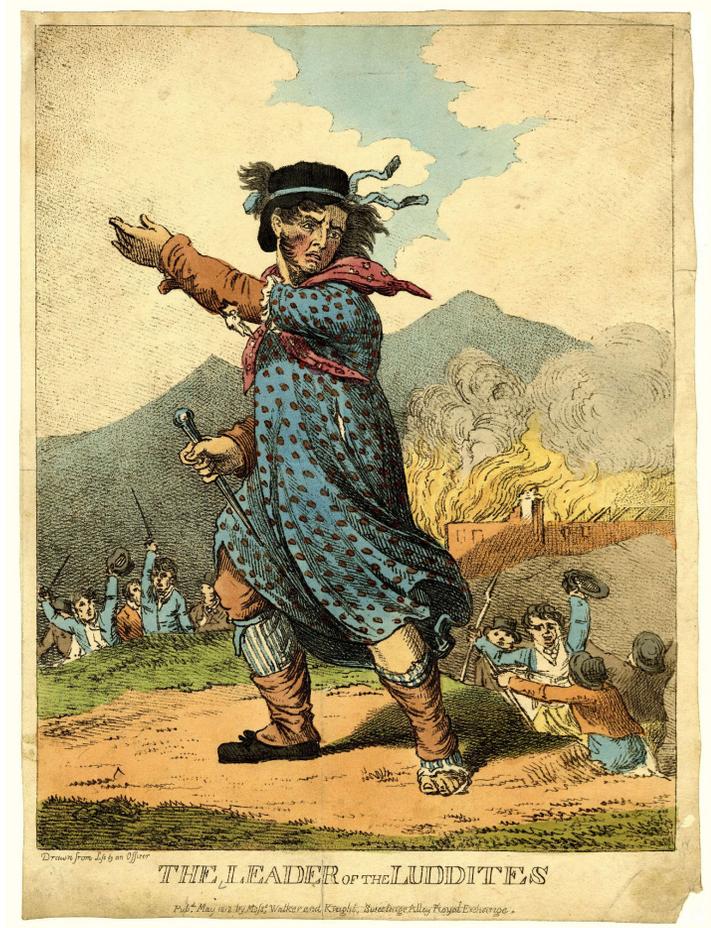
IMP

<https://www.theatlantic.com/ideas/archive/2025/10/ai-deskilling-automation-technology/684669/>

# Is software development being deskilled?

## A very connoted word

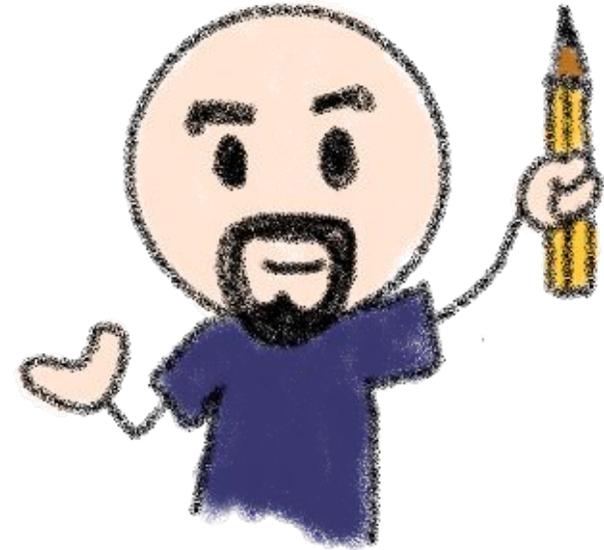
In economics, deskilling is the process by which **skilled labor** within an industry or economy **is eliminated by** the introduction of **technologies operated by** semi- or **unskilled workers.**



# Automation doesn't deskill people

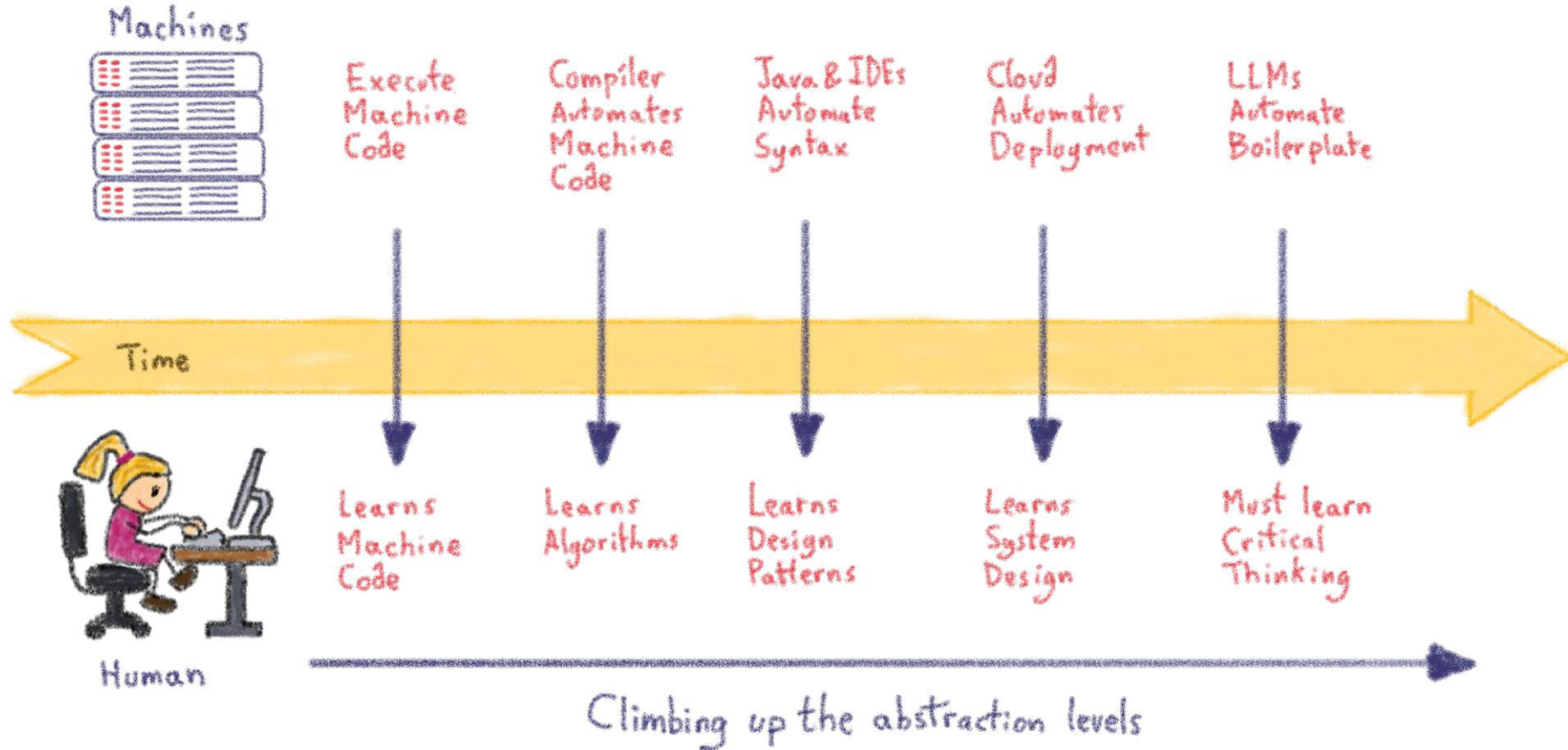
It shifts expertise to places the tools can't reach

Every technology shapes our mind.  
They don't eliminate skill,  
it relocates to a new domain.



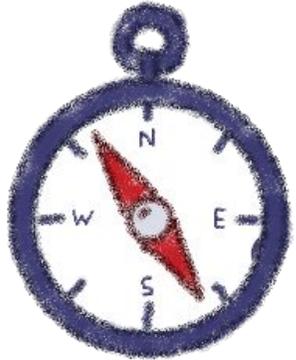
# From Repetition to Reasoning

Automation shifts the skill, not the value



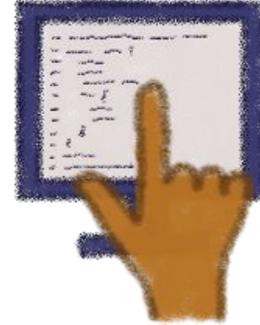
# What We're Really Learning Now

## New literacies for developers



### Framing

Turning intent into precise prompts



### Critical Reading

Validating AI output



### Debugging abstractions

Tracing errors you didn't write



### Ethics & Trust

Knowing when not to automate

# From “Lost Skills” to New Ones

We’re not forgetting, we’re evolving



***“We’ll forget how to code”***

The danger isn’t forgetting how to write a loop, it’s forgetting how to think about one

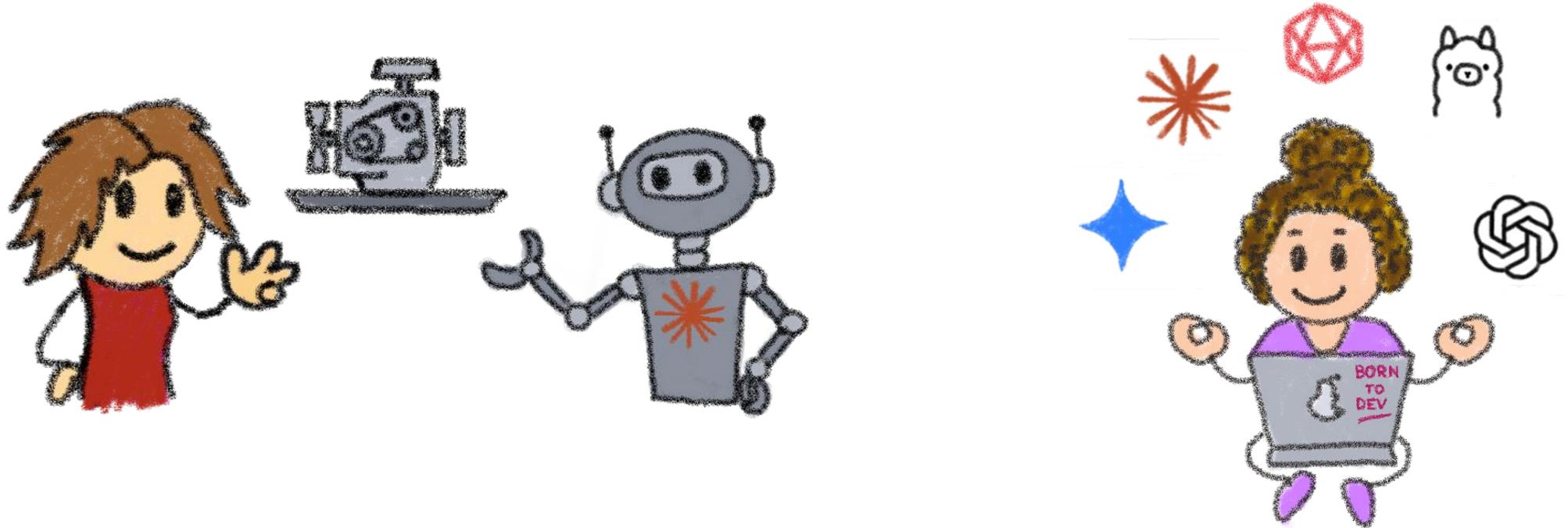
Every machine embodies a social decision, what gets automated, and what remains a skill



Dr. David F. Noble

# The Craft Endures

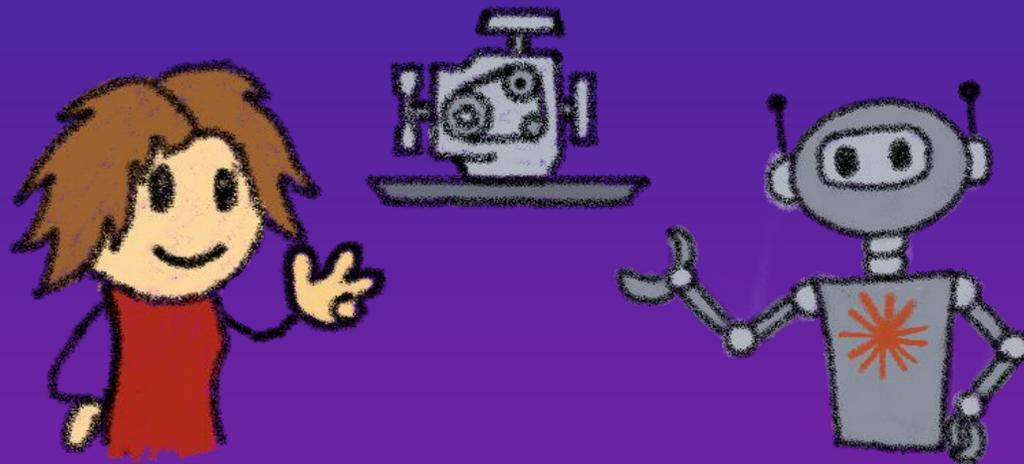
Still human, just differently skilled



When tools learn, so must we  
Automation doesn't end craftsmanship — it redefines it

# The Developer's New Workflow

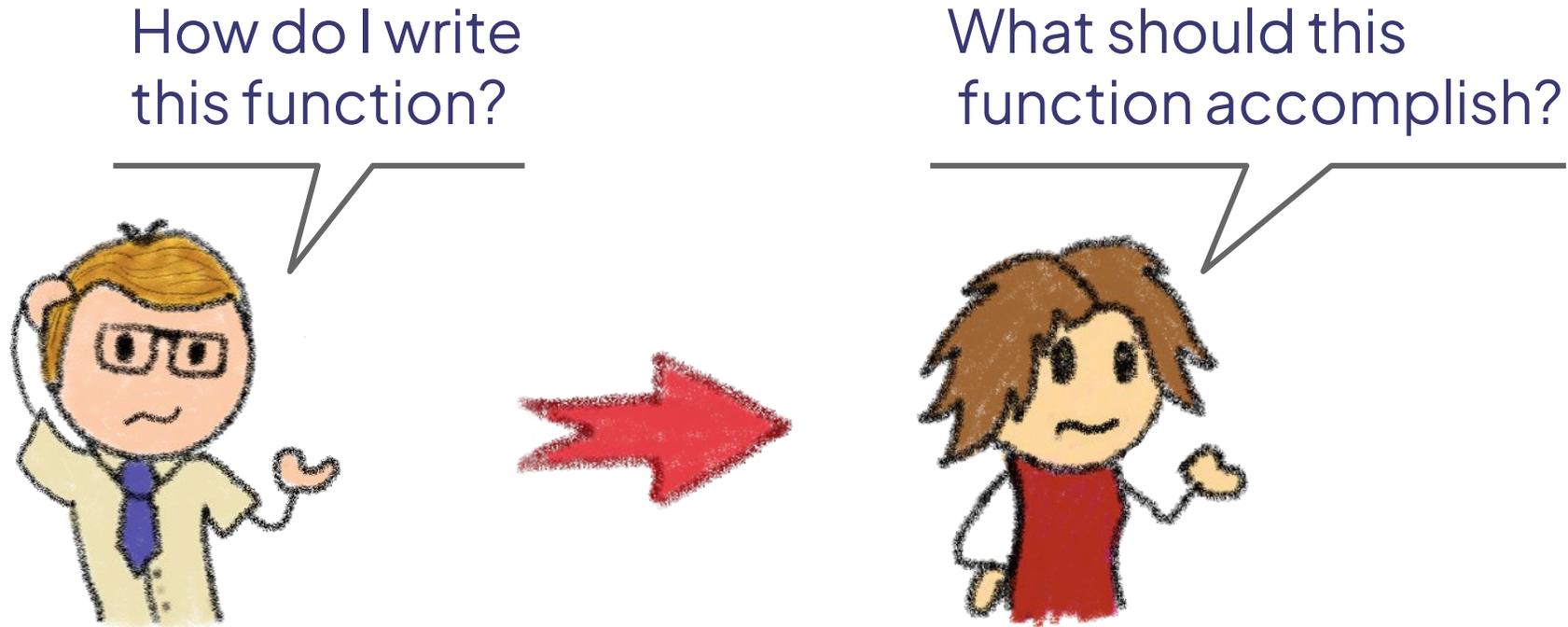
## Co-Authoring with the Machine



# From syntax recall to intent articulation



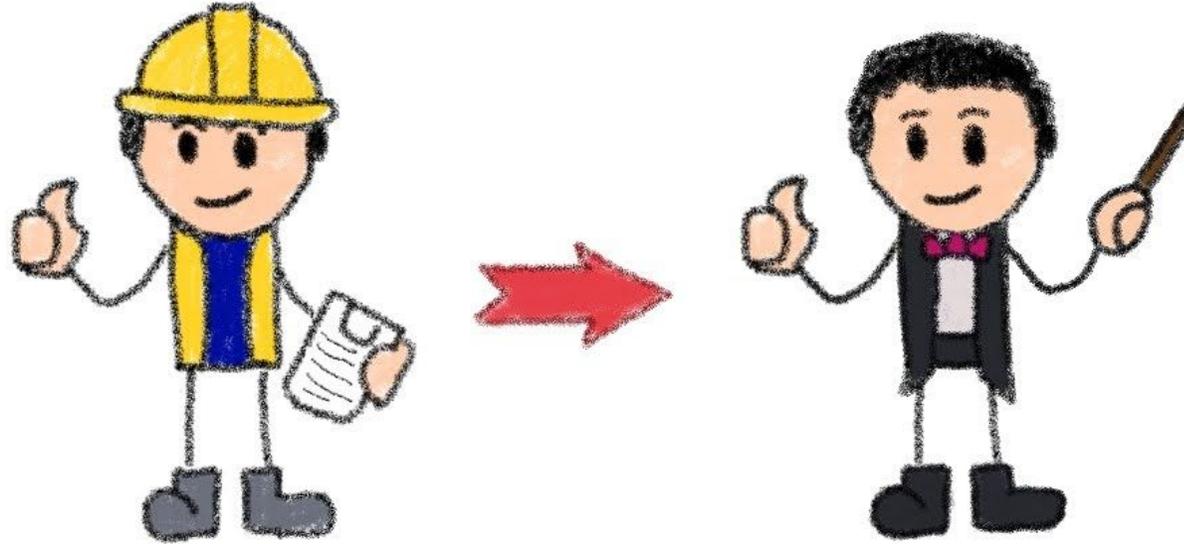
The bottleneck moves from syntax to semantics



You're no longer coding for the machine;  
you're negotiating with it

# From implementation to orchestration

You used to be a builder, now you're a conductor

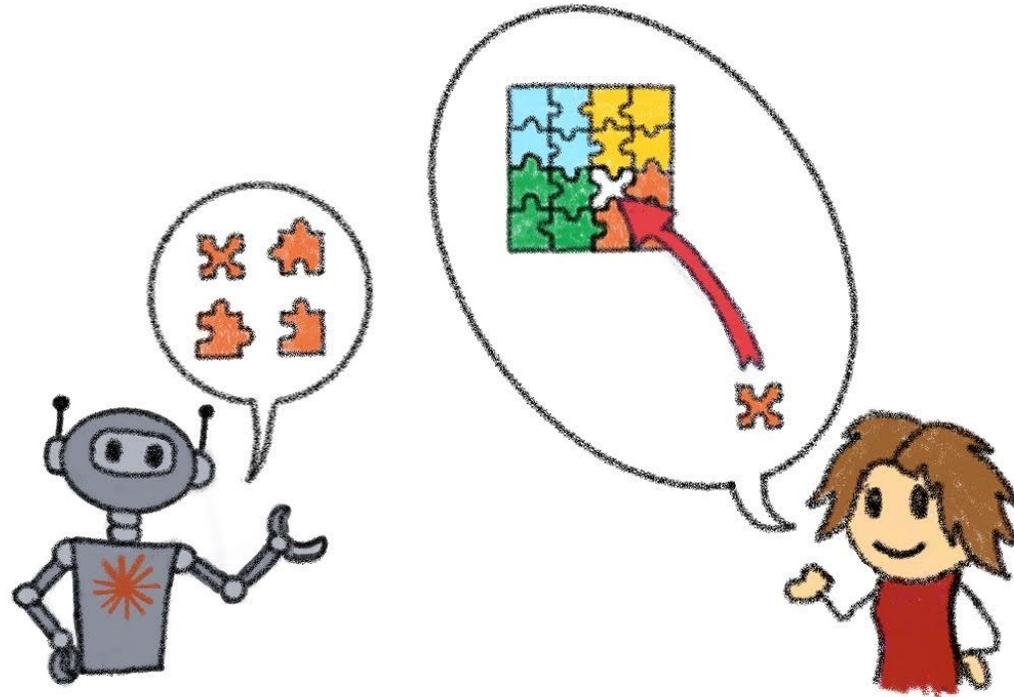


A conductor doesn't play every instrument;  
they ensure harmony and timing

# From writing code to curating systems



Deciding which lines matter, and which ones can be delegated.

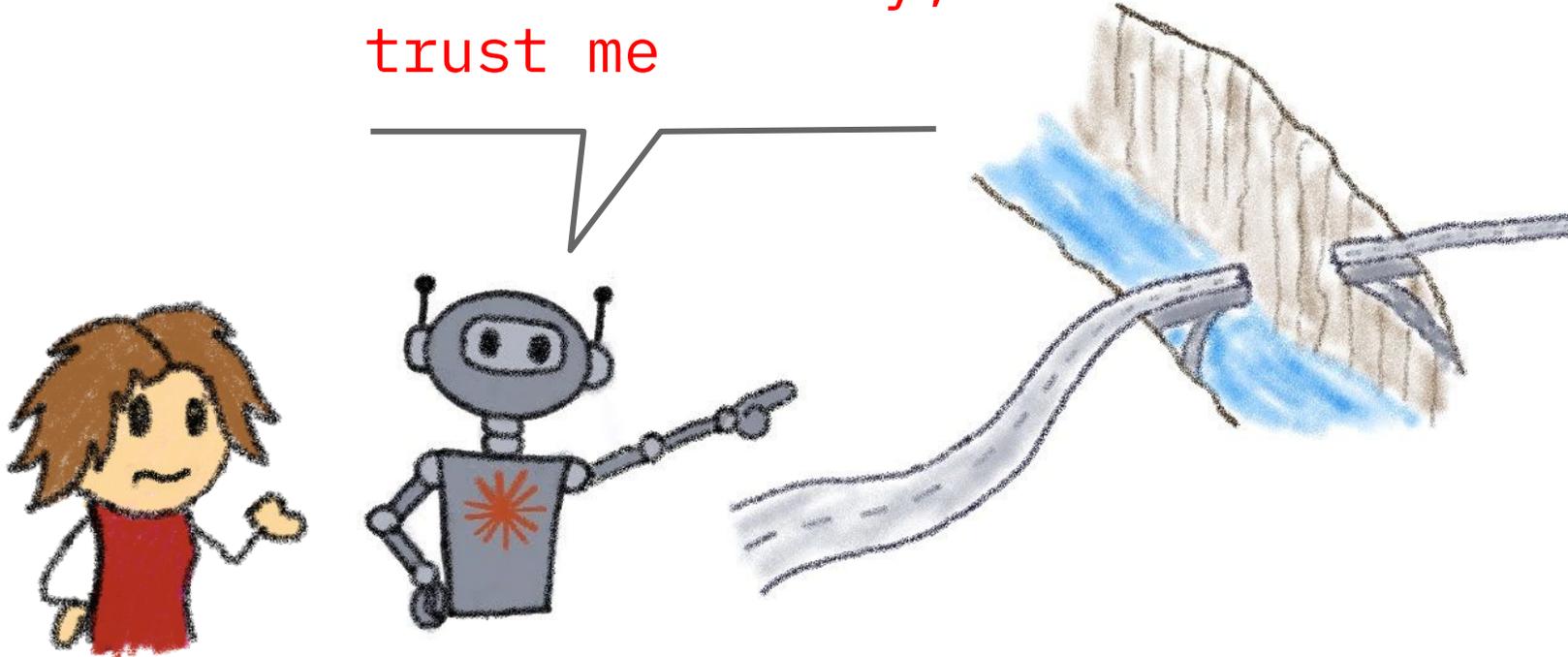


Choosing what to keep, refine, or replace

# Pitfalls

Overtrust, hallucination, loss of mental model

This is the way,  
trust me



The risk isn't that the model will write bad code,  
it's that we'll stop understanding the code it writes

# Vibe Coding: When You Stop Looking at the Code



The word of the year that proves our point

WORD OF THE YEAR 2025  
COLLINS ENGLISH DICTIONARY

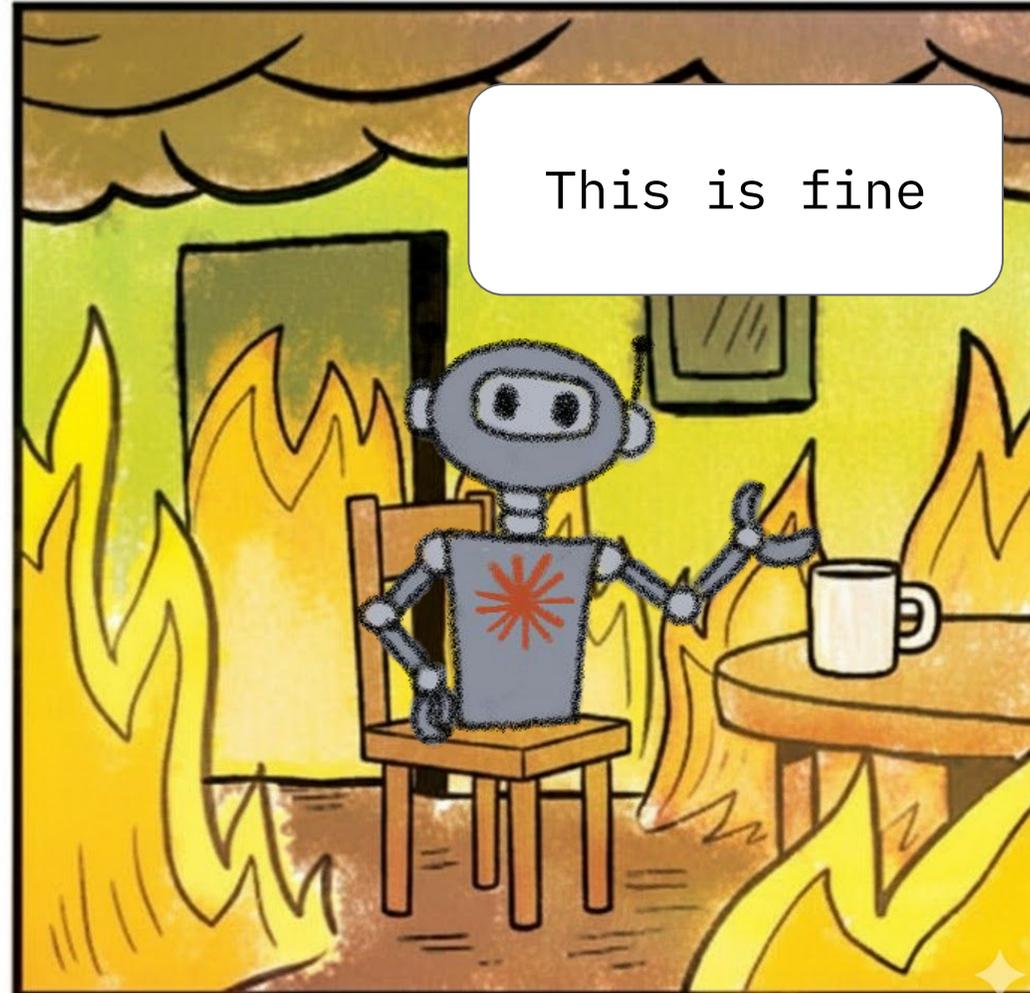
Vibe coding:

Fully give in to the vibes, embrace exponentials  
and forget that the code even exists

— Andrej Karpathy, February 2025

# The AI Slop Crisis

When code works but nobody understands why



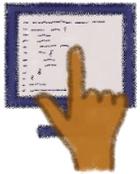
# The new rhythm of collaboration

## Partnering with an LLM

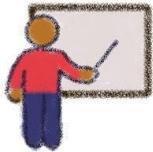


**Ask clearly**

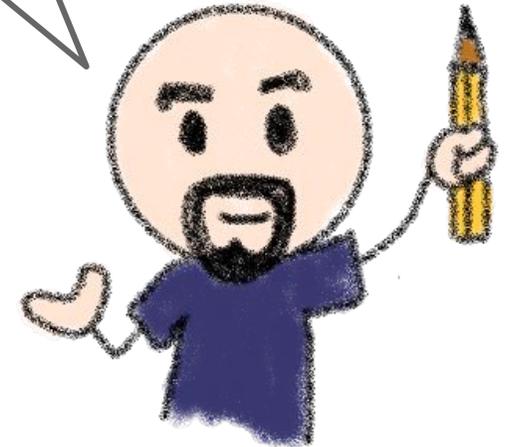
The best developers I know don't treat the model as magic, they treat it as a junior teammate who learns through feedback



**Review ruthlessly**



**Teach continuously**



Coding with an LLM is pair programming with a young colleague who's brilliant, tireless, and occasionally delusional



# From co-authoring to Agent Orchestration



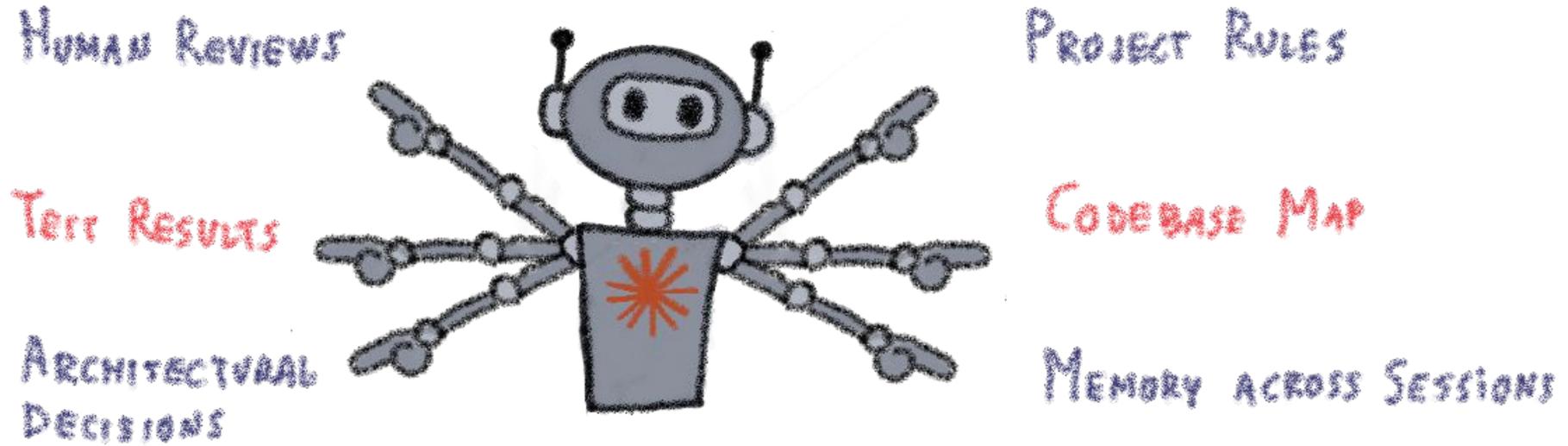
Agentic coding is a reality today



# Context Engineering > Prompt Engineering



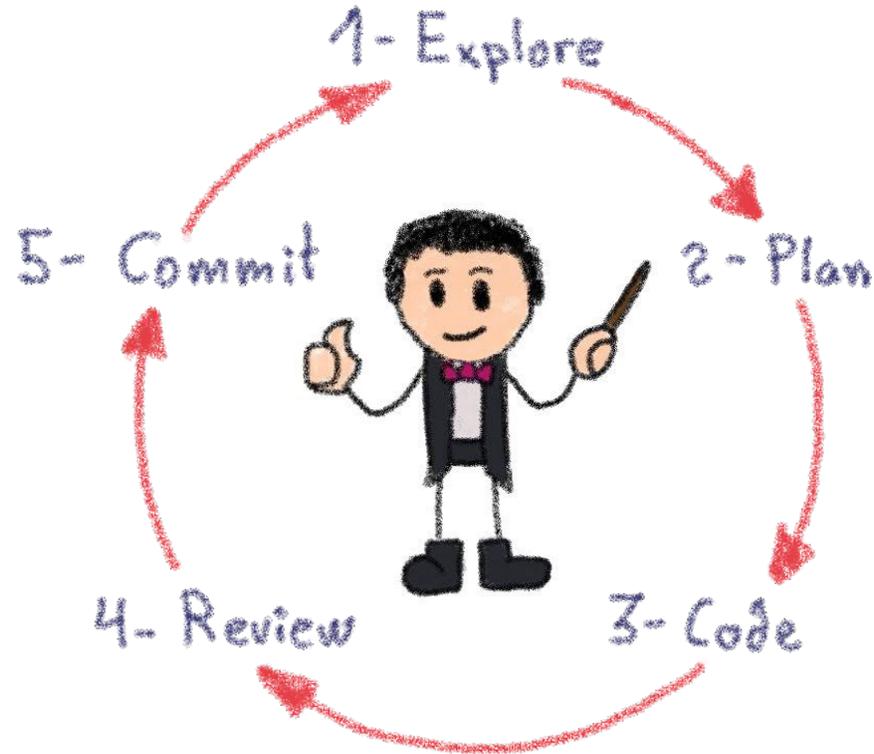
The real skill is what you feed the agent, not what you ask it



The challenge isn't getting models to write code, it's ensuring they see the right information at the right time

# The Professional Pattern

How experienced developers actually use agents



Professionals Don't Vibe, They Control

# In conclusion

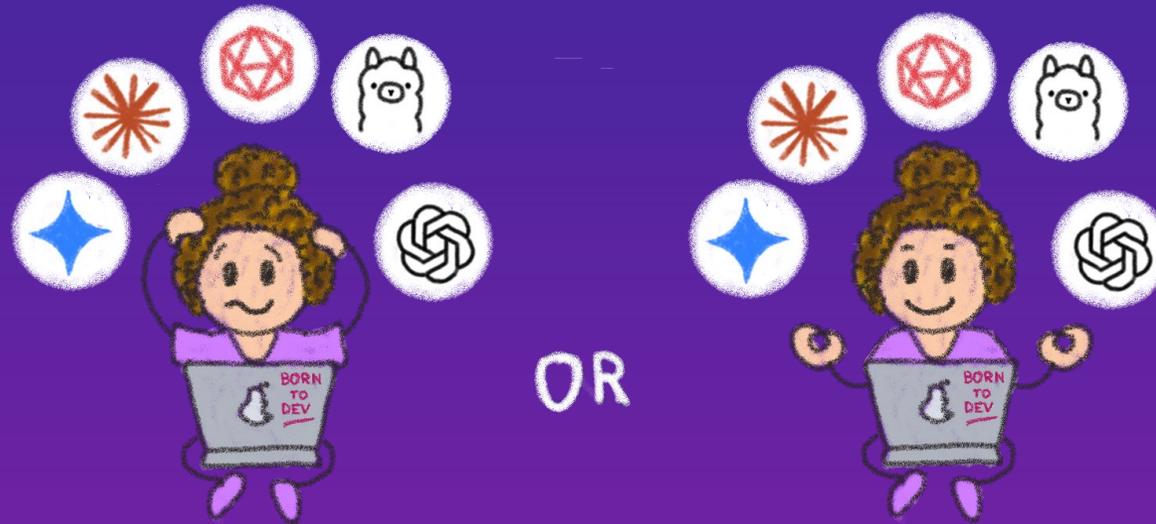
## Conducting an Agentic Orchestra



We used to talk about “writing software.”  
Now we’re talking about “conducting software.”  
The tools play the instruments,  
but we still write the score

# The Developer's Journey

## Growing Up with Smarter Tools

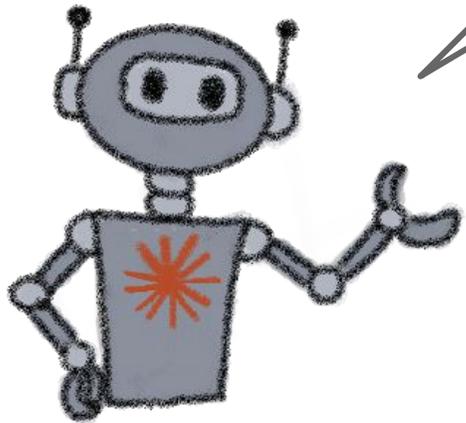


# The vanishing entry-level

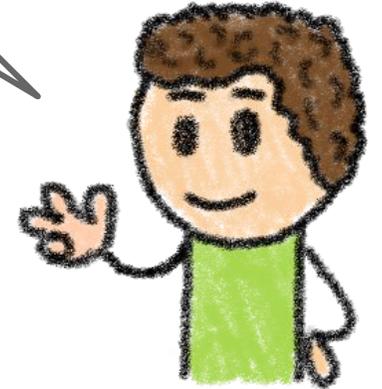


## Setting the stage

I do the scaffolding, the repetitive work, the easy bug fixing, even the commits and PRs



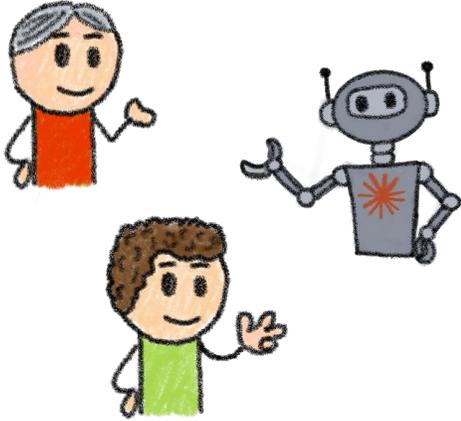
So how can I learn and get better if you do all the easy tasks?



If the easy problems are gone, where do new developers cut their teeth?

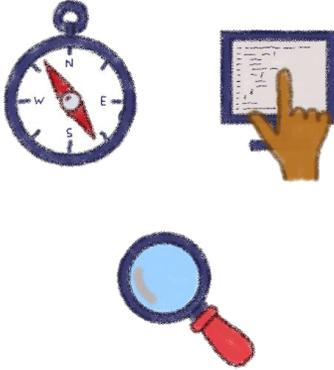
# Rethinking learning

Juniors now must learn through AI, not before AI.



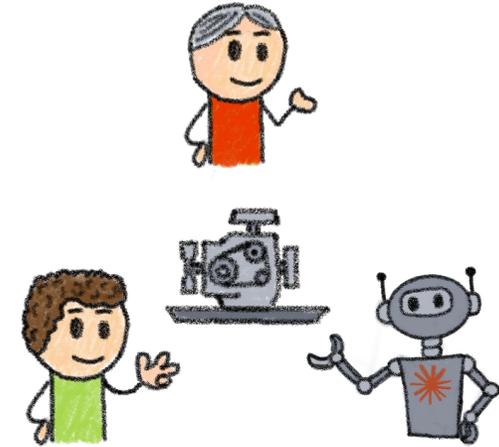
## Tri-programming

Junior, AI and senior



## Teaching

Prompt crafting, critical code reading and debugging AI output



## Guided co-creation

New onboarding pattern instead of rote implementation

# Redefining seniority

What means being senior in a world with AI?



**Senior  $\neq$  years of  
syntax mastery**

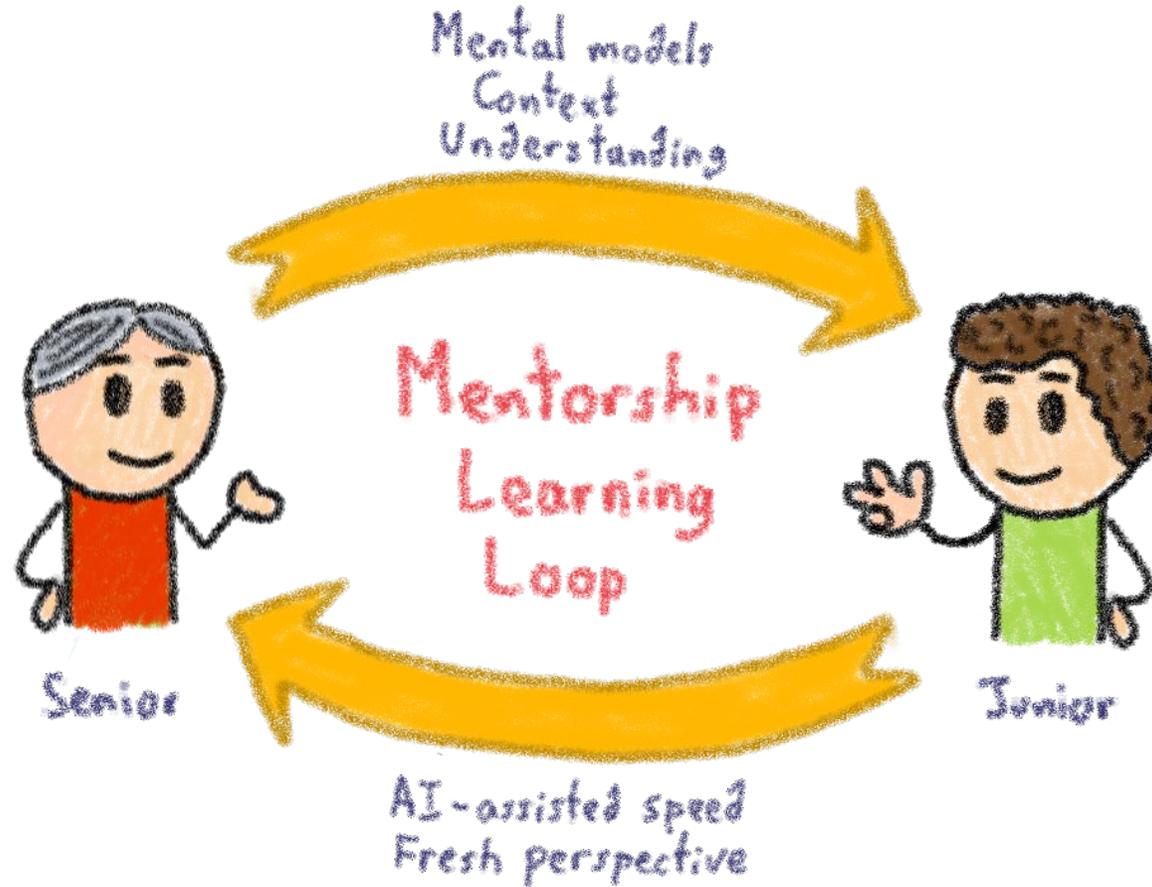


**Senior = common sense,  
system and domain understanding,  
empathy and leadership**

Seniority is shifting from knowing the answers  
to knowing which questions matter

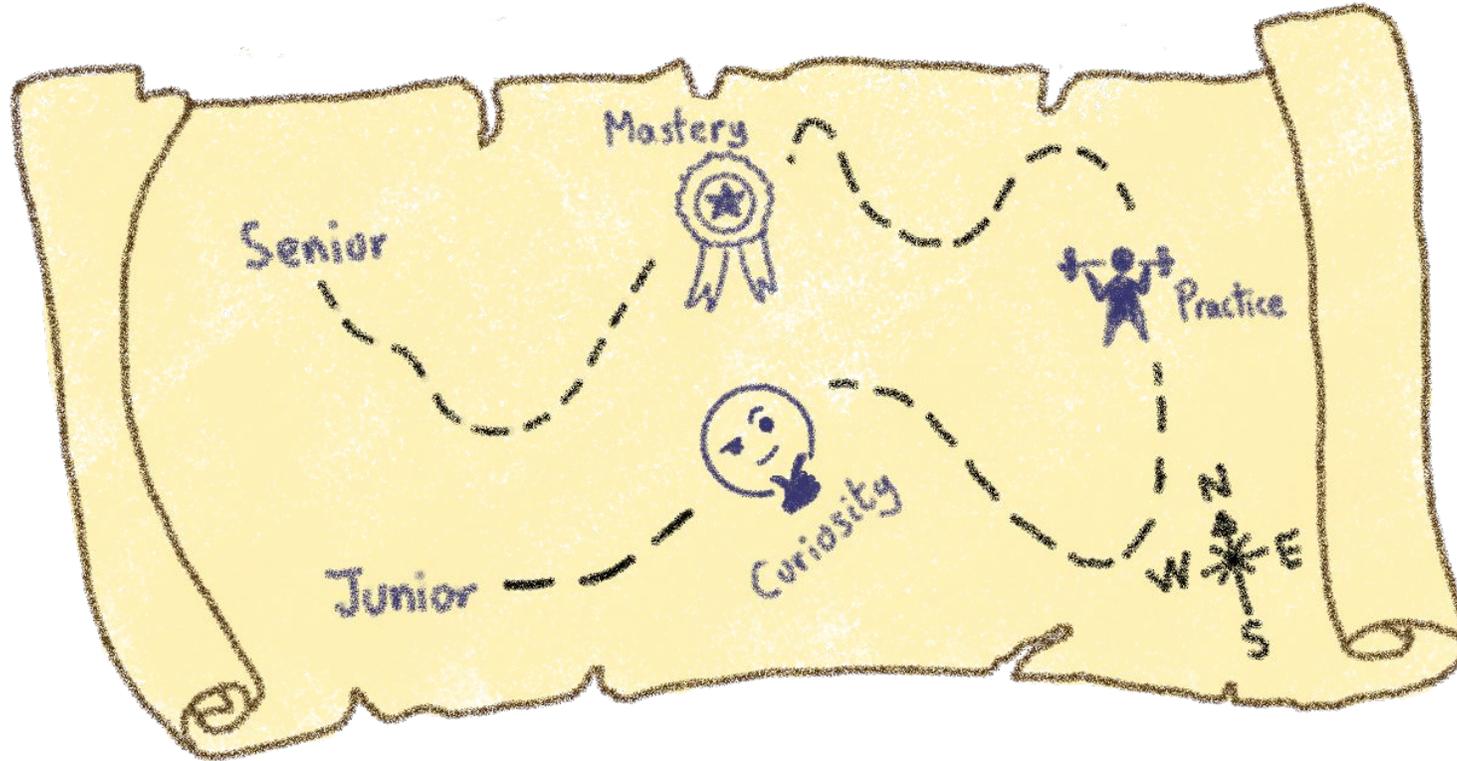
# Mentorship in this new world

A two-ways road



# Conclusion

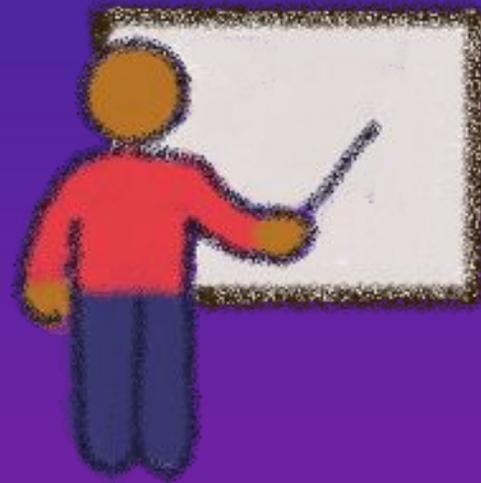
The map changed, but not the destination



Still learning to talk to machines, the language just evolved

# Teaching the Next Generation

How do we teach programming when the computer can already code?



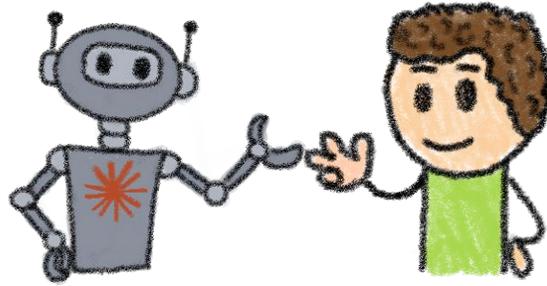
# The broken model

We've been teaching how to code, not how to think about code



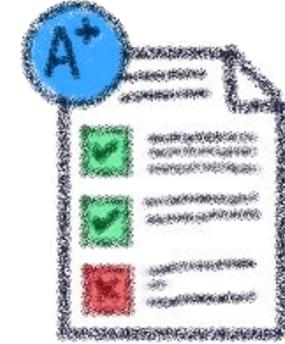
## Traditional model

Syntax drills & algorithmic exercises



## AI assistants

Students can “solve” everything instantly



## Grading output

Everyone can cheat... or worse, learn nothing

# Shifting from execution to understanding

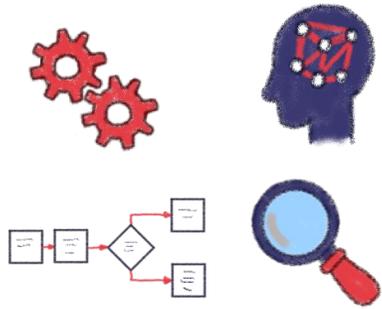


Let's teach less syntax, more synthesis

Write a function that...



Explain what this function does and why



## Change focus

Reasoning, mental models, system design and debugging

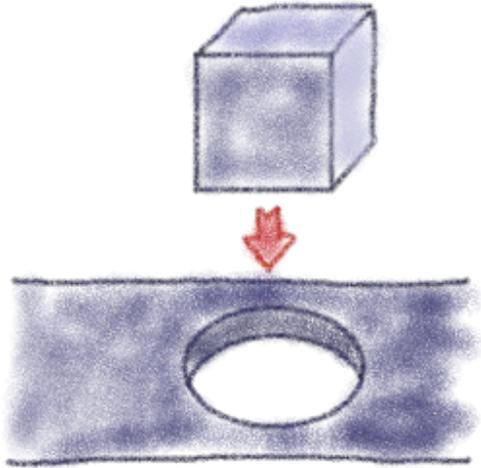


## Evaluate process

Oral defense, live reasoning, code walkthroughs

# The role of friction

We must design friction on purpose



Constraint-based learning forces students to think.

- debug broken AI code
- critique different answers
- give incomplete requirements

The struggle is where understanding grows.

# Teaching collaboration with AI

Students need to learn to use LLMs well, not to hide them

## Prompt design

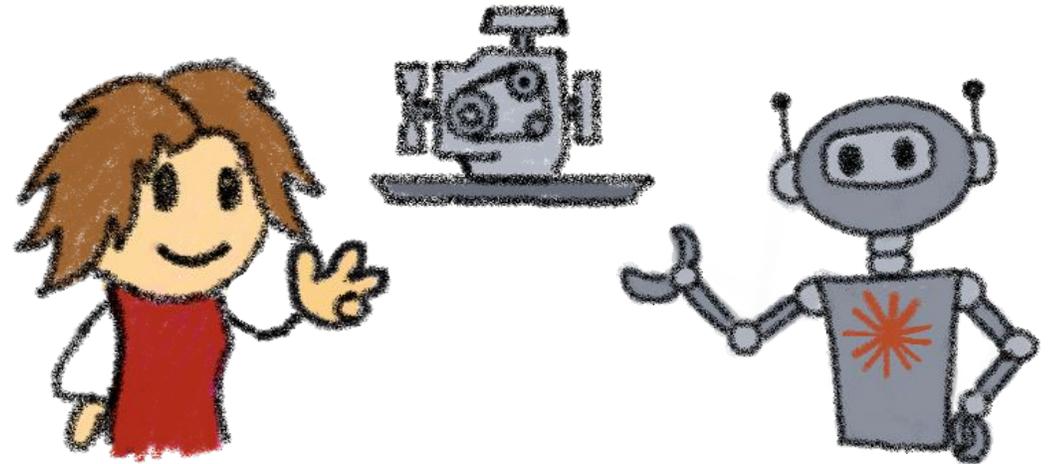
Describe intent precisely

## Verification

Test and analyze the output

## Reflection

Document what was learned  
and what went wrong



# Assessment reimaged

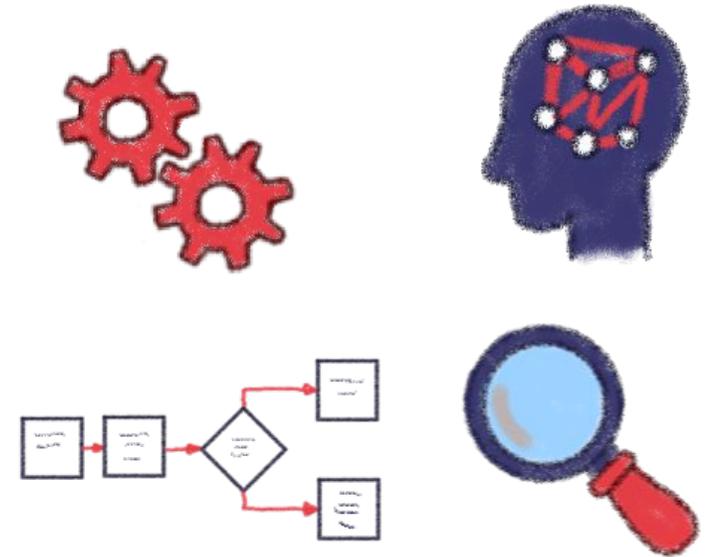
## How can we evaluate?

Plagiarism detection is meaningless

Grade something else:

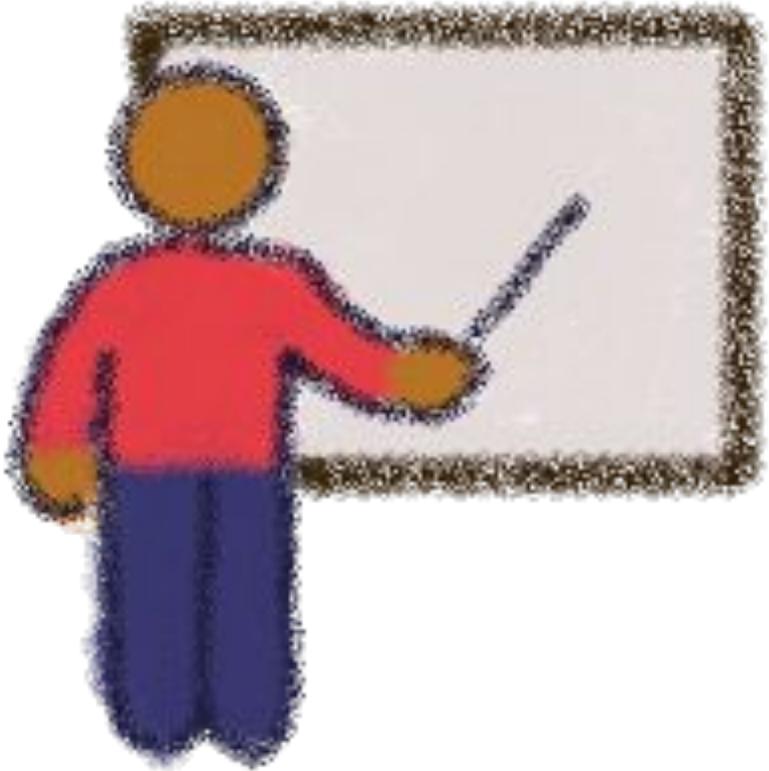
- Process
- Reflection
- Reasoning

When understanding becomes visible,  
cheating becomes pointless



# Re-tooling educators

## Teachers need their own upskilling



We have to learn what these tools do, where they fail, and how to guide students through them

- Experimenting
- Sharing open lesson plans
- Accepting that “teaching AI-era programming” is itself a new discipline

# Conclusion



We don't teach people to out-code the machine.  
We teach them to understand, guide, and question it.

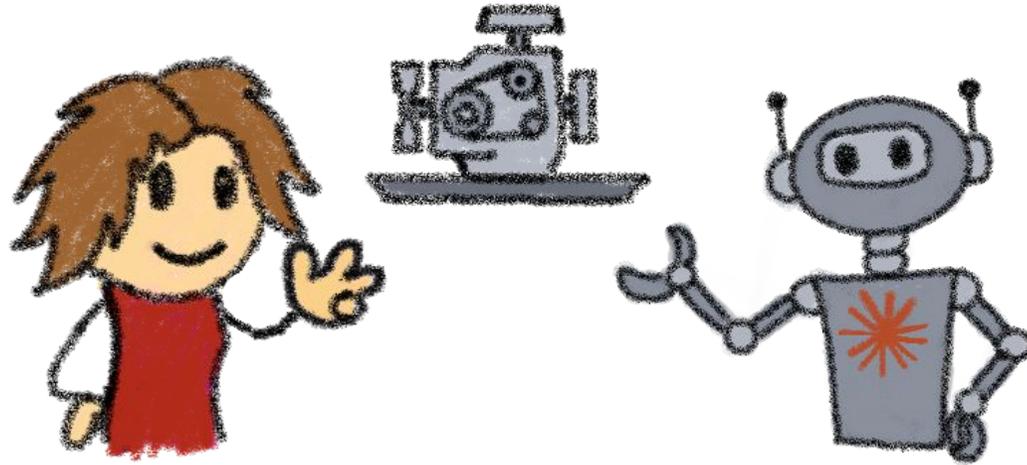
# Differently Human

## The Future of Software Development



# Differently Human

## The future of software development



Every abstraction hides a machine...  
and reveals a human choice.

# The Pattern Repeats

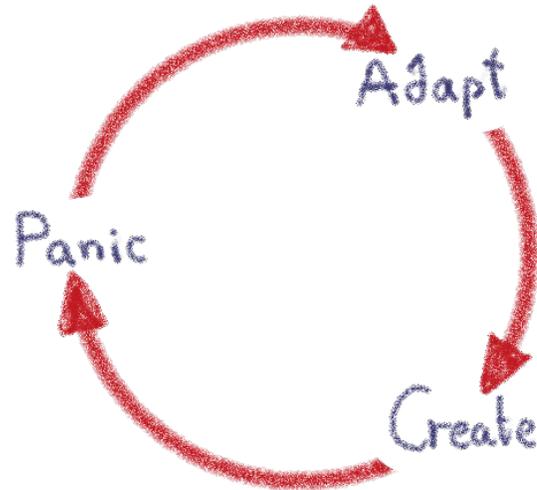


Every revolution ends in rediscovery



Assembly → Fortran → Java →  
Cloud → LLMs

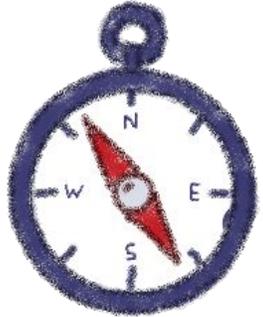
Each looked like an ending  
None erased us; all redefined us



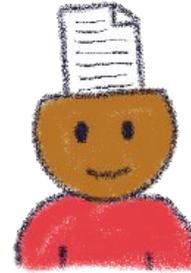
The next wave will do the same

# What Machines Still Can't Do

Computation isn't comprehension



**Intent**



**Context**



**Empathy**



**Ethics**

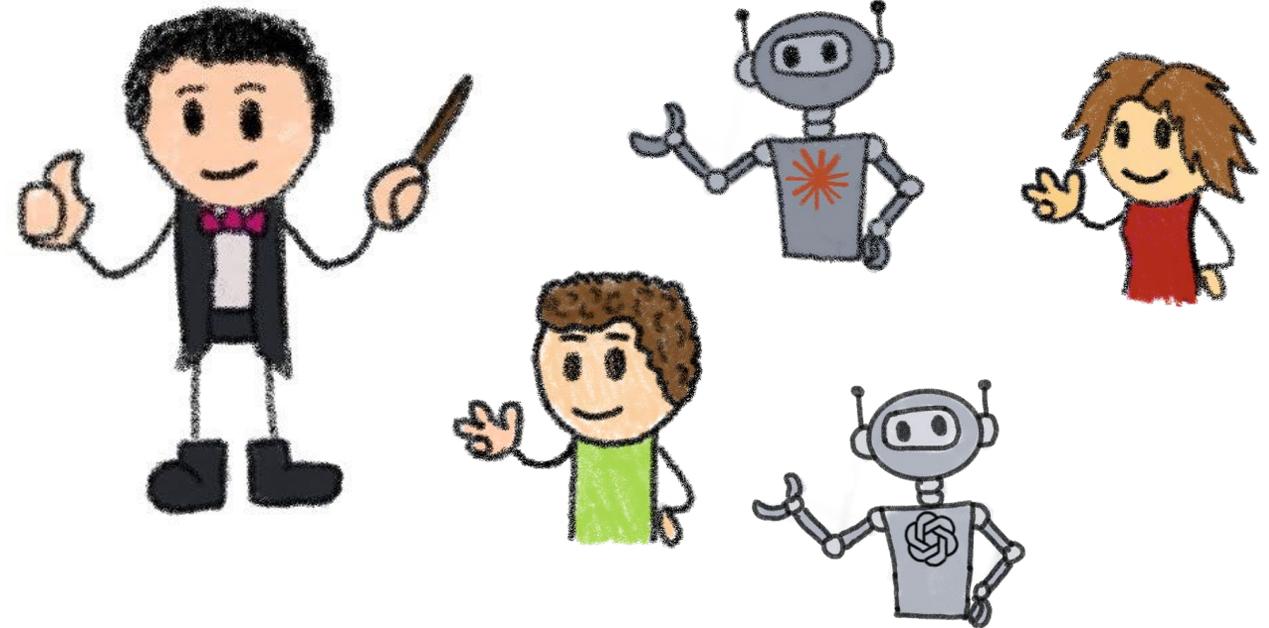
LLMs manipulate form, not meaning

We supply the “why,” the value judgment, the connection to real people

# The New Developer Archetype

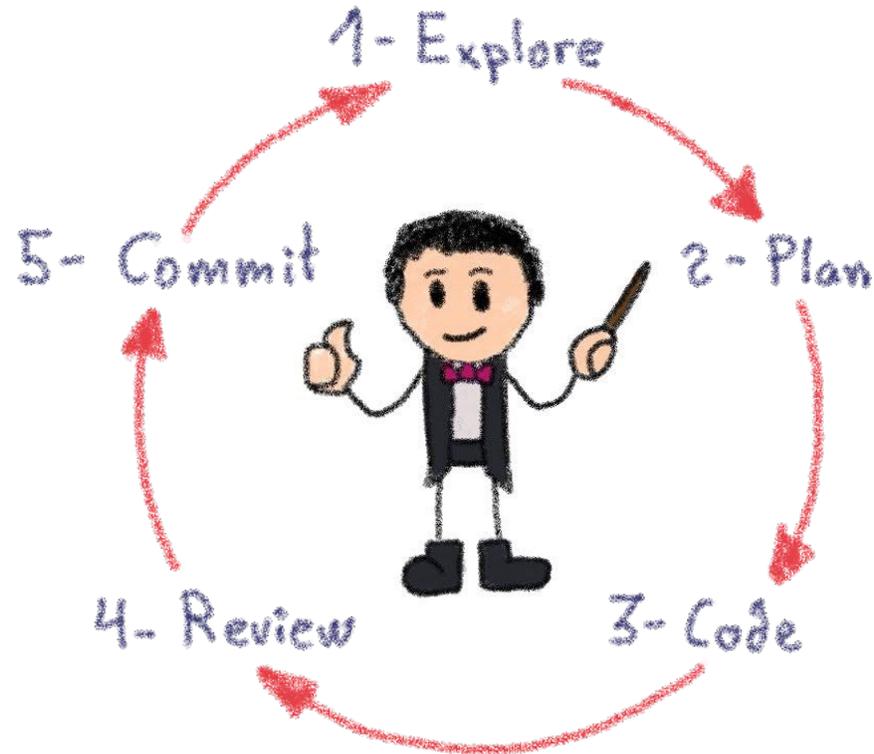
## From Coder to Composer

- Orchestrates human + machine collaboration
- Balances automation with accountability
- Designs systems and stories



# The Human Loop

Keep the Human in the Loop



Our job: preserve understanding inside automated pipelines  
Automation without comprehension is abdication

# Conclusion

## Not Less Human – Differently Human



The future of software development isn't less human.  
It's just differently human.

Our craft remains – it just moves up a level.

# That's all, folks!

## Thank you all!

