Modernisering van Legacy-projecten met AI

Koen Vanderkimpen
Smals Research

4 Juni 2026

Management summary & key takeaways at the end

Aanbevelingen
Conseils spéciaux Webex

Gelieve uw micro op “**mute**” te zetten.
Veuillez mettre votre micro sur “**mute**”.

Gelieve uw webcam **uit te zetten**.
Veuillez **éteindre** votre webcam.

Problemen met het geluid? Klik op “Audio&Video”, “Switch Audio ”, “Disconnect” en reconnect via “Use Computer Audio’.

Problèmes de son ? Cliquez sur “Audio&Video”, “Switch Audio ”, “Disconnect” et reconnectez-vous avec “Use Computer Audio’.

Problemen met het headset? Klik op “Audio&Video”, “Speaker and Microphone settings”, doe de testen met het juiste type headset.

Problèmes de casque ? Cliquez sur “Audio&Video”, “Speaker and Microphone settings”, faites les tests avec le type correct de casque.

Vragen? Stel ze in de groepchat (Everyone). We behandelen ze aan het einde van de webinar.

Des questions ? Posez-les via le chat de groupe (Everyone). Nous les traiterons à la fin du webinaire.

Modernisering van Legacy-projecten met AI

Smals Research — Koen Vanderkimpen

Donderdag 4 juni 2026

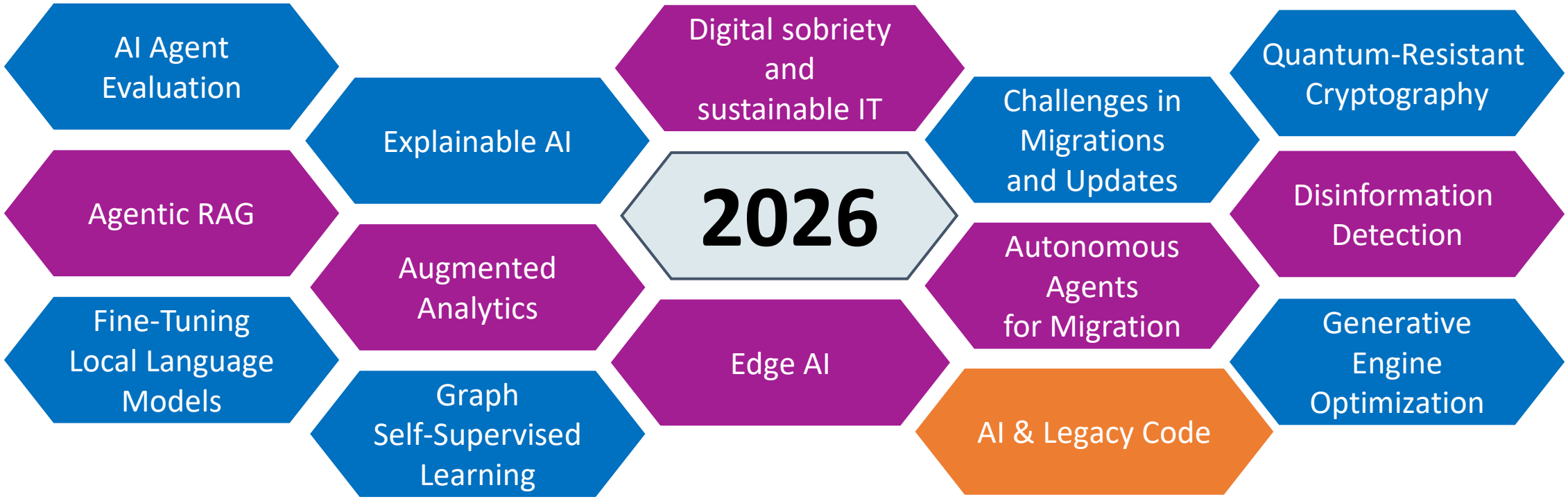
Smals Research 2026

Innovation with new technologies

Consultancy & expertise

Internal & external knowledge transfer

Support for going live



www.smalsresearch.be

Agenda

- **Introduction**
  - Very short history
  - AI & Development: broadening the Scope
  - Harness Engineering
  - The Legacy Spectrum
  - AI for Legacy, specifically
- 6,5 Cases
- Cases: Preliminary Conclusions
- Zooming out on AI Development & the Future
- Conclusions - lessons learned

Smals
ICT for society

Dec 2023: webinar AI Augmented Development

(by my colleague, Joachim Ganseman)

- Avoid overreliance and beware of results
  - AI is overconfident
- Best results with small amounts of work
- Very good for the typical project scaffolding ↔ Will write very typical code in typical frameworks
- ***"soon, coding without assistants will feel like writing without spellchecker"***

And my "Vibe Coding" blog, anno 2024:

- 'Using AI will keep working only at small scale'
- Out of reach: library replace, module pull-out, large refactors

Anno 2026?

This is no longer an evaluation of a technology

This is learning how to best leverage it

(knowing all too well the technology is continuously outpacing us)

Smals
ICT for society

AI & Code: From Chatbots to Vibe Coding & Beyond: a mere 3y evolution

You could ask a **Chatbot** your question about the code, OR …

→ **Code completion** (“smart” became “AI-powered”)

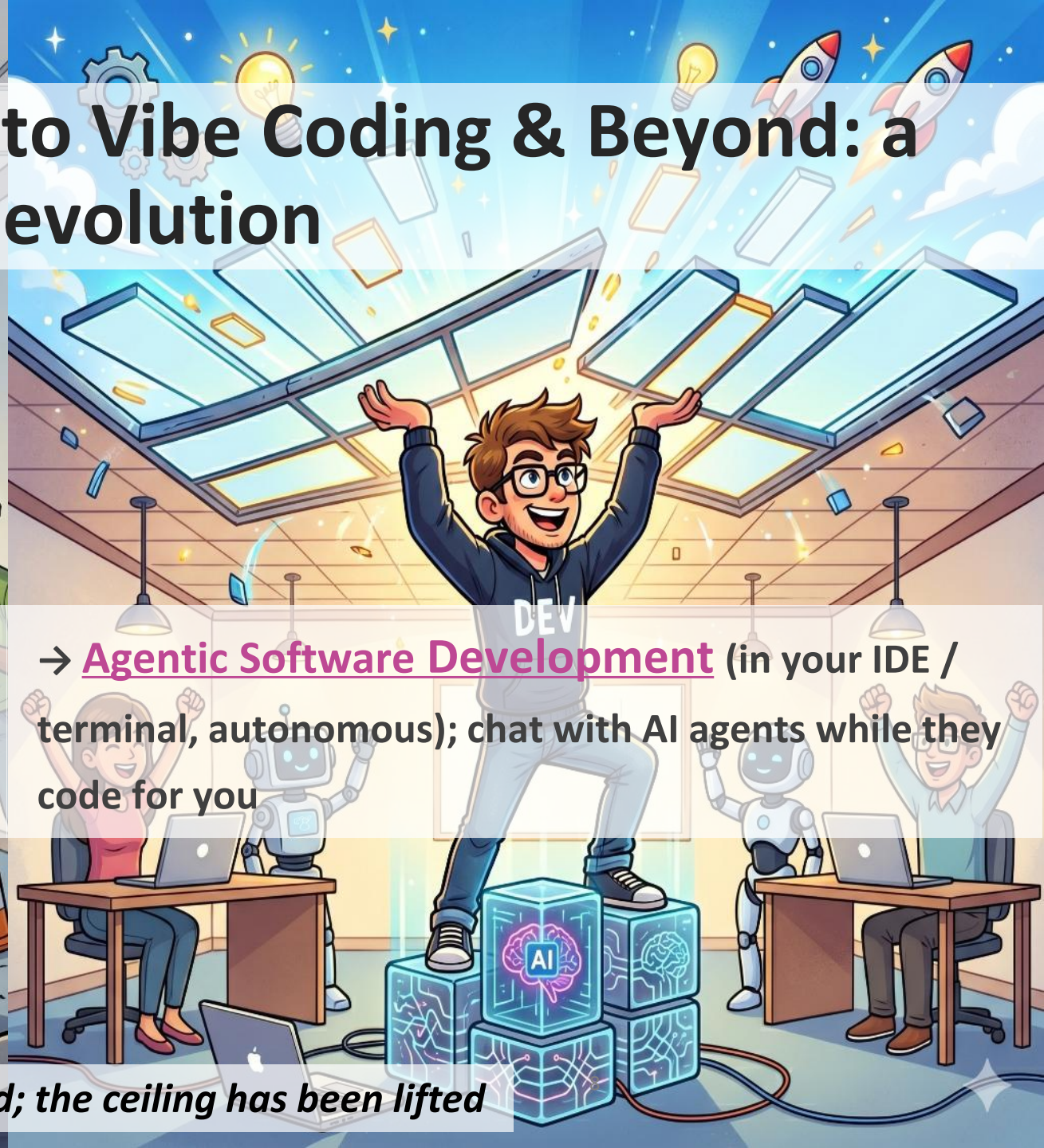
→ **AI coding assistants** (LLM in your editor, writing short pieces and assisting in-file)

→ **Agentic Software Development**

→ **Vibe coding**

Smals
ICT for society

AI & Code: From Chatbots to Vibe Coding & Beyond: a mere 3y evolution

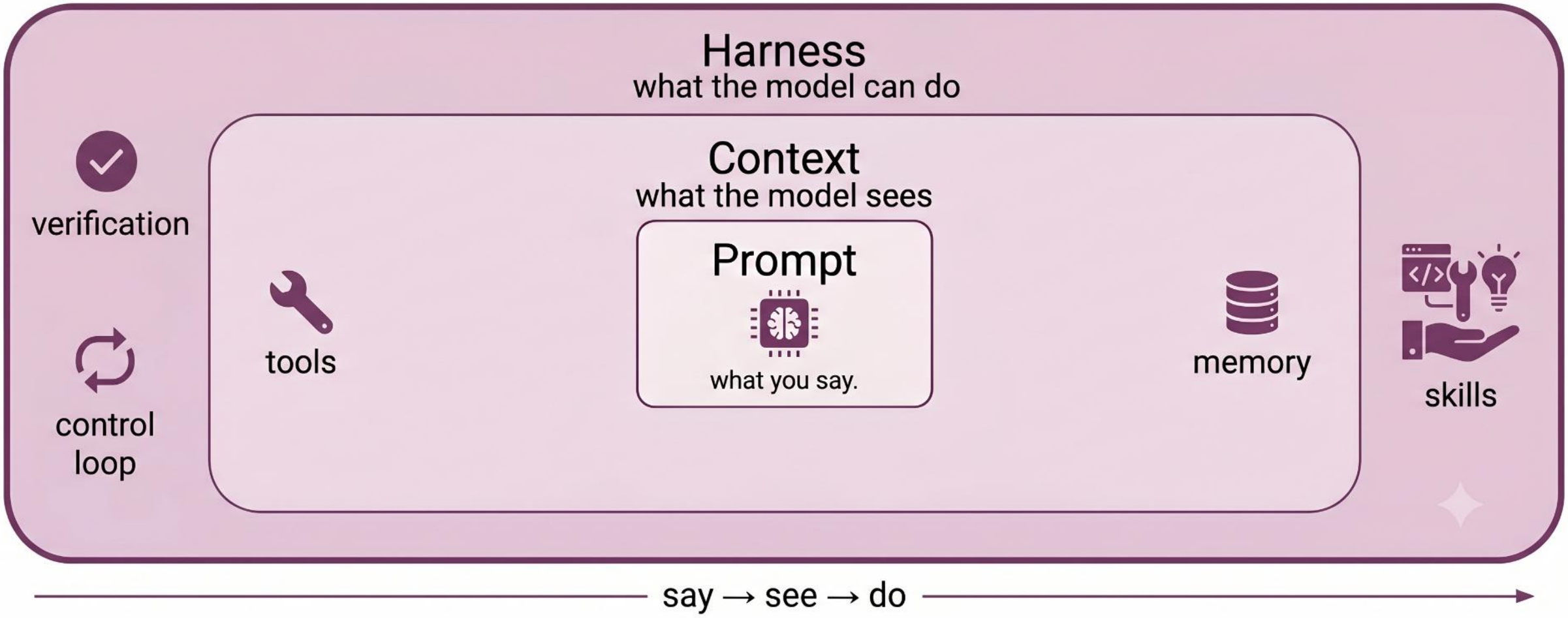


→ Vibe coding (talking to a system that writes an application for you; no skill required)

→ Agentic Software Development (in your IDE / terminal, autonomous); chat with AI agents while they code for you

The floor has been lowered; the ceiling has been lifted

Agentic Coding: towards Harness Engineering



"Plan, Execute, Verify"

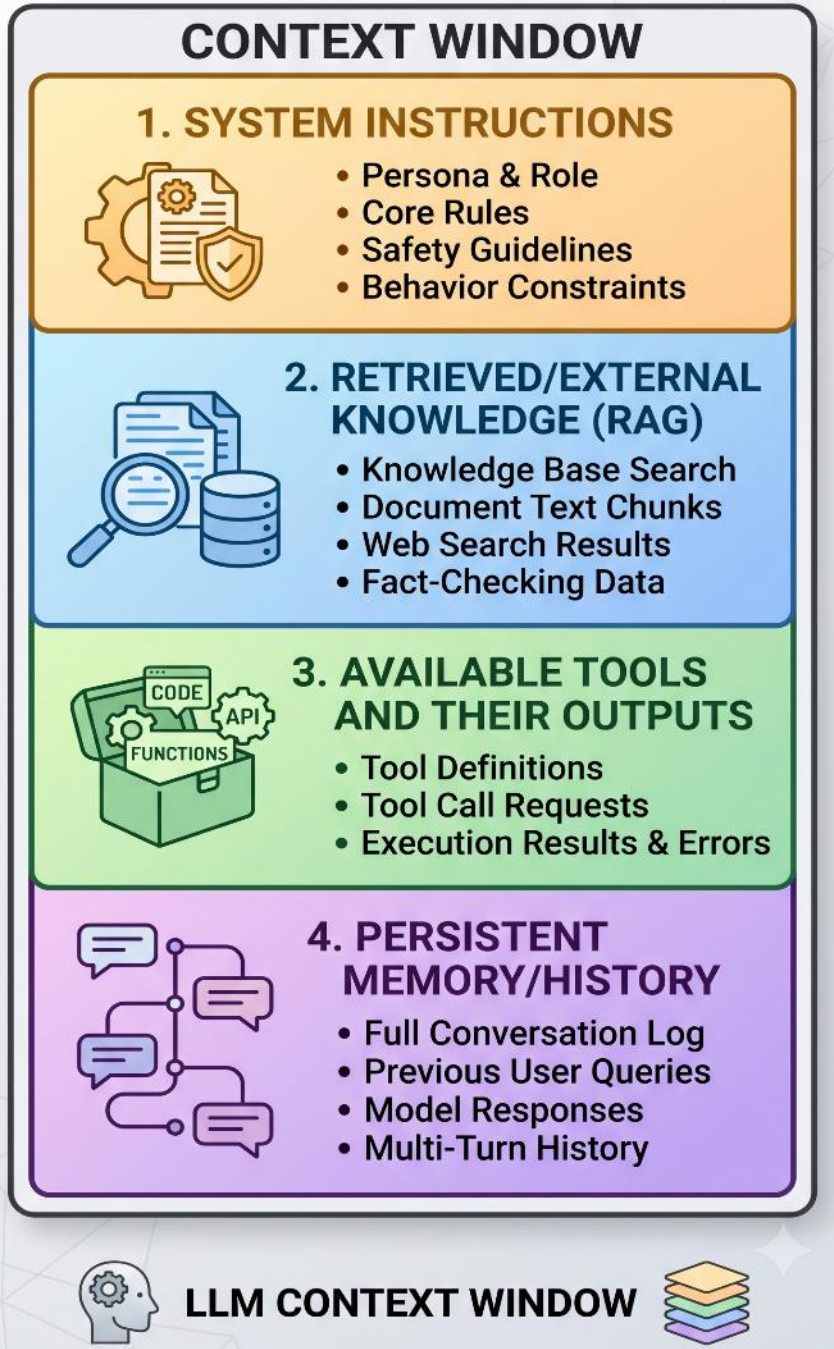
Smals
ICT for society

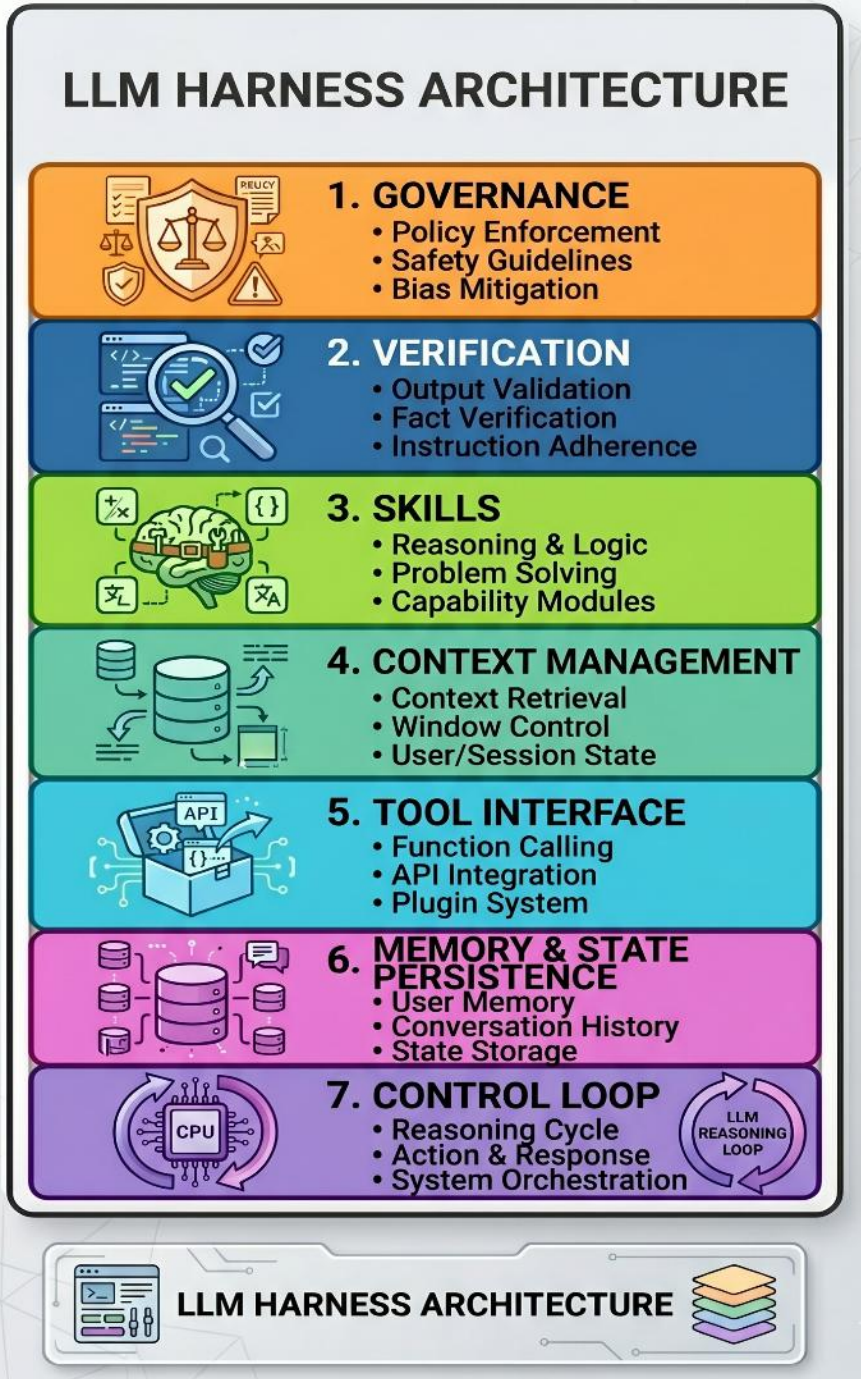
Context is King !

Context engineering = managing the entire context window, not just writing prompts

- **prompt** : *what you say to the model*
  ↔ **context** : *everything the model sees*
- More isn't better
  - irrelevant or excessive context degrades performance
  - ("context rot," distraction, conflicting signals)

Context engineering = curating the *minimum high-signal set* of tokens for the task





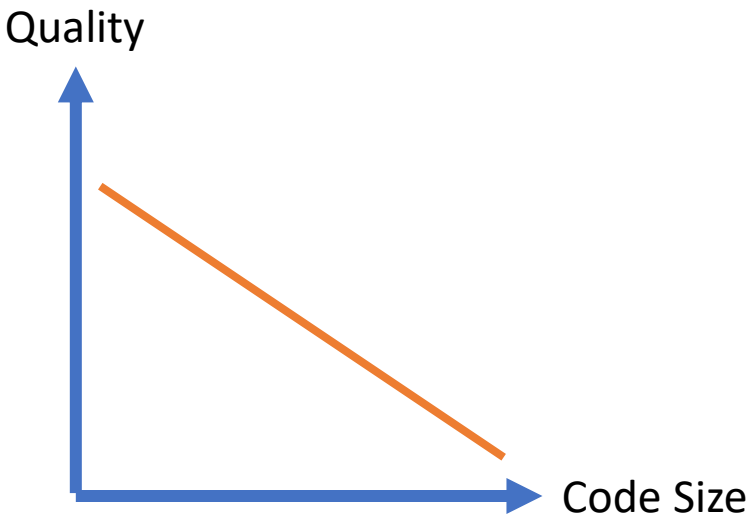
Harness is the Kingdom !

Harness engineering = context + infrastructure; the whole agent system

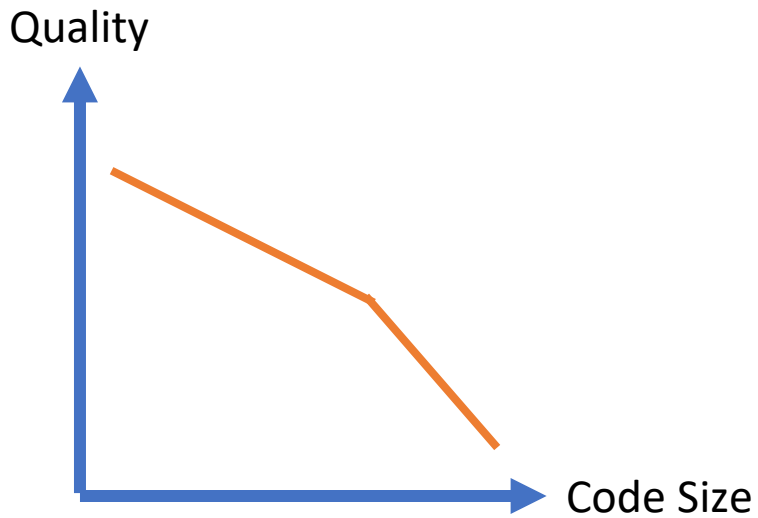
- **harness**: *everything that wraps around the model*
  → optimizes everything the model can do
- A capable model isn't enough
  - **reliability** depends on the system wrapping it
  - E.g. skills introduce repeatable recipes → consistency
  - failures are system problems to fix, not prompts to retry

Smals ICT for society

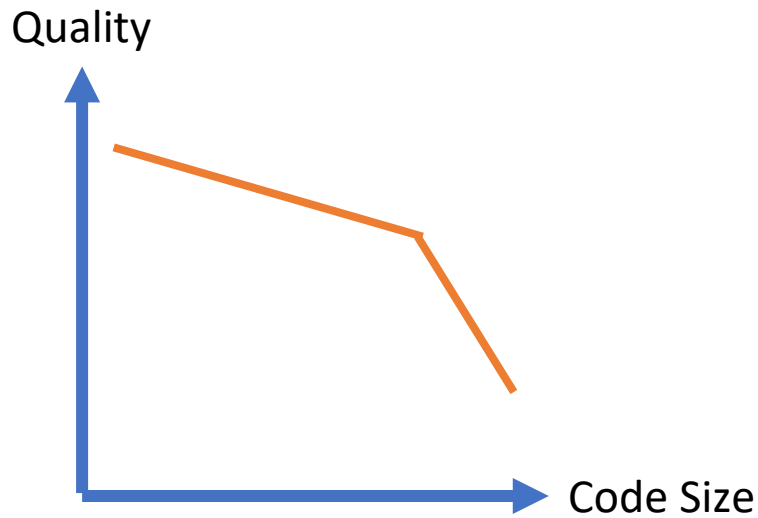
Harness Engineering vs previous (my take)



Prompt Engineering



Context Engineering



Harness Engineering

Smals
ICT for society

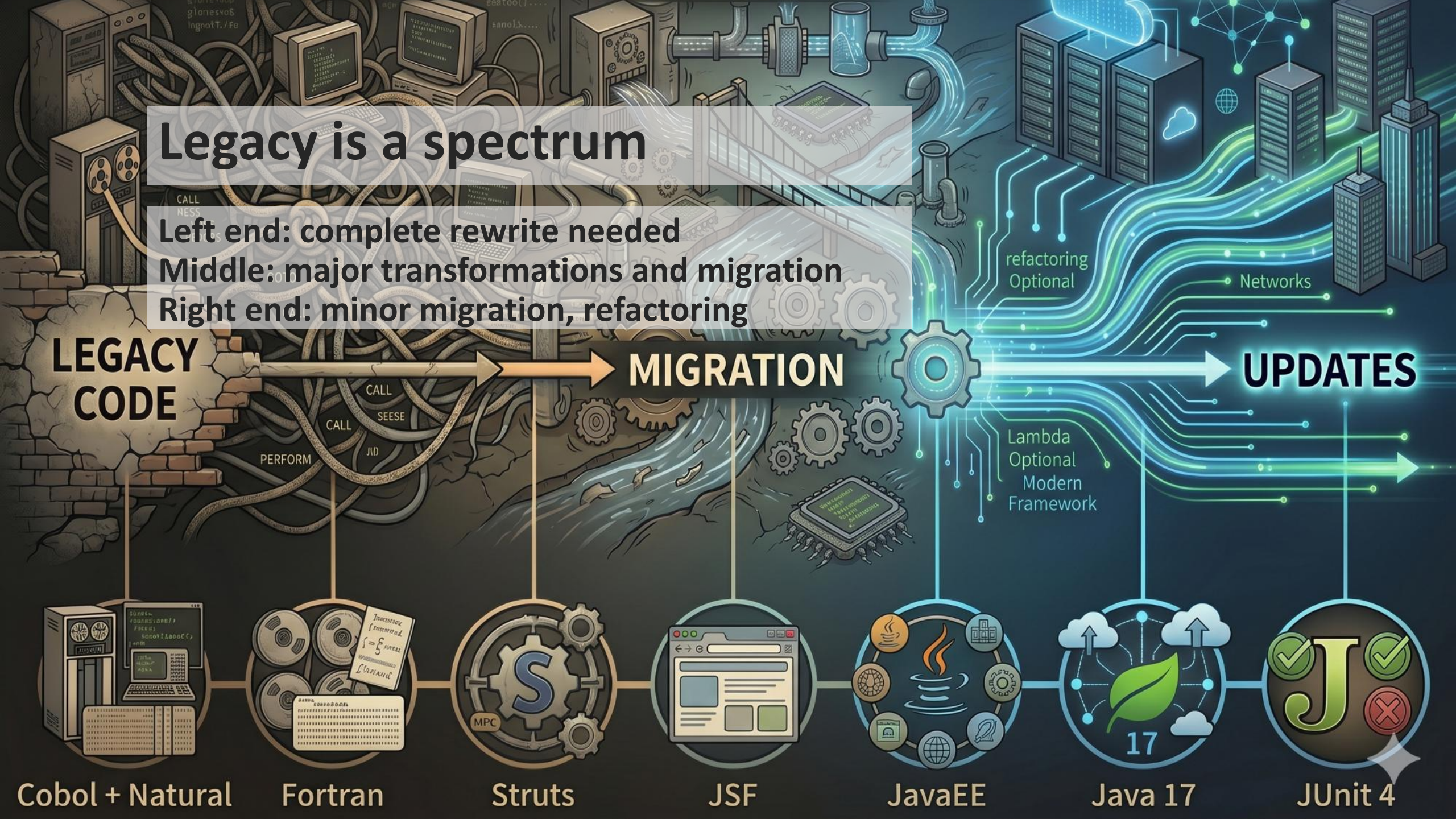
What is Legacy Code?

(No real formal definition)

Why does it usually exist? Cost to maintain ↔ **Cost to change**

Legacy is a spectrum

Left end: complete rewrite needed
Middle: major transformations and migration
Right end: minor migration, refactoring



AI & Legacy Code ⊆ AI & Coding

- Codebase size and the need for new architecture often makes rewriting legacy too challenging to put AI in the driver's seat
- The existing (enormous) context of code is usually too big
  - Either to get completely into the context window
  - Either to create a meaningful, focused context for good results

  → **same as AI assisted coding when the project starts to become very big**
  - Must find creative ways to split, modularize, re-architect, …
  - (But that is often precisely the problem with legacy code)

AI & Legacy Code ⊆ AI & Coding

- But when we do write code, we are (almost) doing the same thing as when writing new code
- Lessons Learned in this study often apply to AI assisted development in general !
- The closer we are to the *migration/update* side of the spectrum, the more often this applies

Smals
ICT for society

AI & Legacy Code ≠ AI & Coding

Main different pathway to handle legacy: analyzing / documenting your old code

- With Legacy, it is sometimes better to get the useful stuff ***out*** of it
- Rebuild something new with that knowledge!
- AI can be useful as “archaeologist”
- Or as “oracle” chatbot

What can AI do for us today?

- AI ≠ LLM — an LLM is one building block (a 'stochastic parrot')

→ Real power comes from agents + harness + context around the LLM

- Good at **searching, analyzing, summarizing, coding**
- Weaker at real problem-solving or complex architecture
- Can leverage tools and skills !
- Beware of reliability
- context drift/rot: **size of codebase matters**

The Spectrum, and our cases

Legacy
complete rewrite

Migration
major transformation

Update
refactor & minor migration

Smals
ICT for society

Agenda

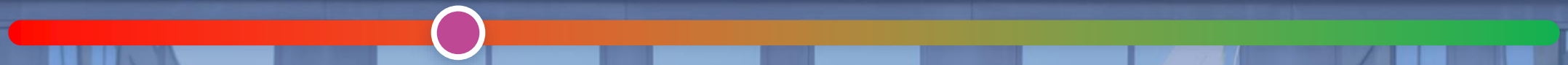
- Introduction
- **6,5 Cases**
  1. Java Rewrite assistance
  2. COBOL conglomerate analysis
  3. COBOL documentation/chatbot
  4. SAS -> Python Migrations
  5. Jakarta Migration
  6. Word Macro Analysis
  - *Starting Up: smaller COBOL; Translations*
- Cases: Preliminary Conclusions
- Zooming out on AI Development & the Future
- Conclusions - lessons learned

Smals ICT for society

1. Java Technical Rewrite

Human Project in Progress

Can AI help speed it up?



Smals
ICT for society

A Java Technical Rewrite— problem and approach

- Belgian eHealth sector
- Old Java code (java 8), with old frameworks in use (e.g. Struts); becoming difficult to maintain; modernization wanted
- end 2024 - begin 2025
- Technical (isofunctional) Rewrite already decided
- How can AI help?

→ Using AI Agentic Coding, can we implement a feature in the new project, while reading the business logic in the old code?

Smals
ICT for society

Java Technical Rewrite Assist— what worked / what didn't

✅ Correct code to implement the feature was found in the old codebase (after a few tries), including database access instructions

✅ New feature successfully written after a few hours and a couple of corrections

Remember: 2024, one of the first cases; more the era of prompt engineering

⚠️ The feature was not large (<200 loc, but within a much larger context)

⚠️ We had to hold the AI's hand to make sure the correct architectural paradigms of the new project were followed; even so, we still ended up short, at around 80-90 % correct

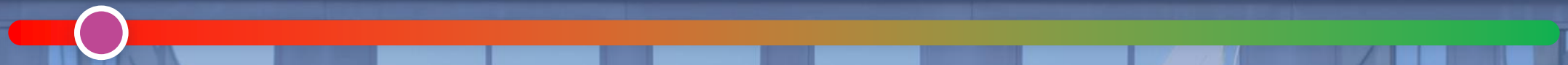
Main takeaway

- AI needs rigorous constraints in the form of a good harness

2. COBOL Conglomerate

Old Collection of Batch Programs; infrastructure no longer supported

Can AI do something?



Smals
ICT for society

A very old COBOL Conglomerate — the problem

- Belgian social security sector
- Stack: RHEL 5 + MicroFocus Cobol ES 5.1 + TPX Telebig + Oracle 10.2 —
  - obsolete + unsupported → unsecure and potentially unstable
  - Very risky to maintain
- Inventory: 454 BATLIB CBL/PCO, 789 CBL+PCO total, 216 JCL, 107 SQL, 1,307 INPUT, 123 Oracle tables

  More or less ☺
- First questions to answer: translate to Java? Emulate? Upgrade to GnuCOBOL?

Smals
ICT for society

COBOL Conglomerate — approach (1)

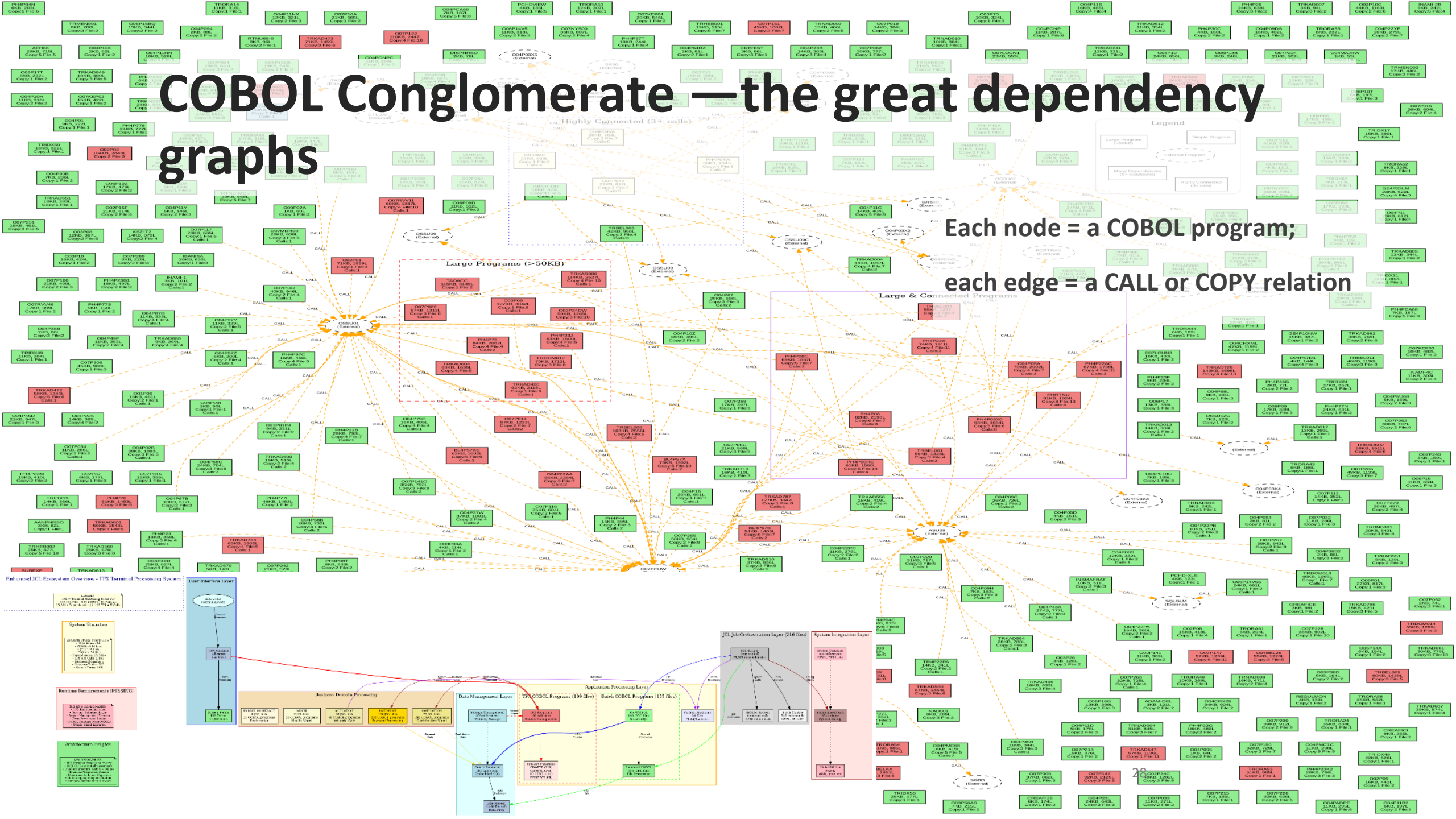
- Phase 1: dependency analysis
  - Claude builds framework of 22 Python analyzers (CBL + PCO + JCL dependencies)
  - Output: JSON + CSV + DOT + Graphviz PNGs : massive spiderweb of dependencies
- Phase 2: COBOL upgrade POC
  - Translate some smaller and less dependent programs to GnuCOBOL, another dialect of COBOL with open source software to compile and execute it.
  - Compile + run + fake input → ELF binary, output look ok
- Problem: this is where it ends
  - No available test server to see if the new programs can integrate with existing stack or correctly handle real input with verifiable output
  - Translation may have mitigated some of the more immediate risks, but ultimately do not modernize this application → it was deemed more useful to re-engineer the conglomerate

COBOL Conglomerate — approach (2)

- Phase 3: AI deeper analysis of the code
  - Problem: without very specific guidance; analysis and advice tend to be generic/bland; not very useful

- Phase 4 (future work, project mode): more analysis of the code by the AI; also scope limitation
  - Need to search for more specific things that are actually useful (e.g. functional documentation)
  - dependency chart a good start of a knowledge base

COBOL Conglomerate —the great dependency graphs



COBOL Conglomerate — what worked / what didn't

✅ Dependency graph gives useful insights into the code, after writing the necessary tools using AI → indirect assistance, building internal toolchains; very useful

✅ Two programs compile and run under GnuCOBOL: easy translations are possible

⚠️ Other static analysis by AI producing technical explainer documents not very useful; business logic extraction may have been better

COBOL Conglomerate — main takeaways

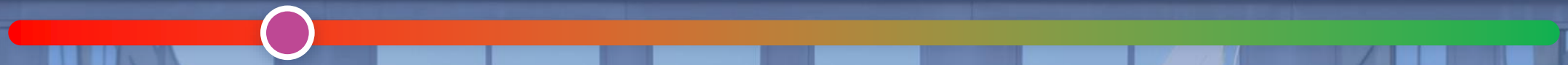
Main takeaway

- AI can do literal translations between programming languages, especially if they are (very) close together
- If AI can’t handle the codebase size; the tools it writes can!

3. COBOL Documentation Case

COBOL code with modern program in the pipeline

Can AI make it easier to maintain the code a few more years?



Smals
ICT for society

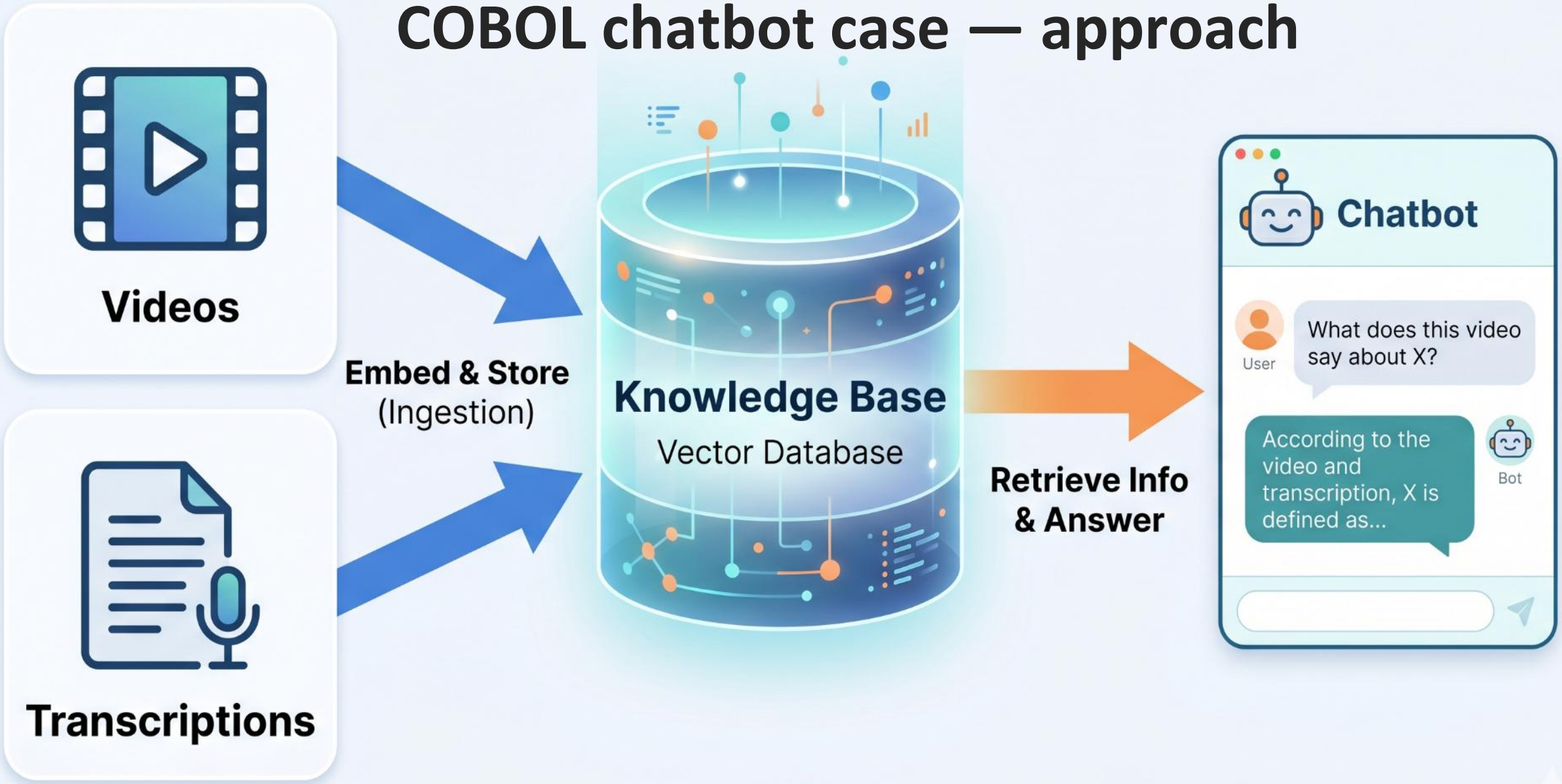
Another COBOL case – Docs or Chatbot ?

- Another social security application — 250+ COBOL programs/files, late '90s / early '00s
- Functional documentation never written
- Lots of technical documentation available (close to the code)
- Key employee retired March 2026; provided explainer movies: videos on what the code does
- Project already started to rewrite the business logic in another, newer application
  - BUT: the COBOL program has to be maintained until that new app is finished (3 more years)
- Hard human deadline
- Preliminary question: better docs? Later on: chatbot!

COBOL chatbot case — approach

- Basic Approach: feed case-specific info to an AI to create a custom chatbot
- Bottom-up: M365 Copilot agent + Sharepoint document corpus (“knowledge” for the agent)
- *The explainer movies as well as AI-generated transcripts from these movies*
- Based on that, create a chatbot, available in Teams, to answer questions about the COBOL program
- Later expansion: include written documentation and sources; still an option
  - And perhaps a dependency analysis would also be a good expansion

COBOL chatbot case — approach



This case as a RAG application — sibling lessons

- This IS a RAG application, à la MSFT: Copilot agent + Sharepoint corpus + retrieval at query time
- Sibling research: Katy Fokou & Bert Vanhalst — 'Generative AI on your own data' (Smals, 7 Nov 2025)
- Their pipeline (Collect → Extract → Clean → Chunk → Embed → Enrich) maps onto this project
- 'Garbage in, garbage out'

COBOL documentation case — what worked / what didn't

✅ Copilot agent published to defined user set; good preliminary test results

⚠️ Improvements may still be possible using the docx and dependencies, and even the code itself

Main takeaway:

- RAG works quite well to improve the quality of a chatbot
- Chatbot can quickly deliver the right information in a specific context; could be better than documentation

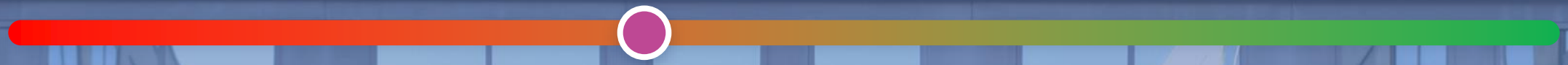
Future work:

- Perhaps the knowledge base can be reused by an agent to help implement Java code ??

4. SAS → Python

Large set of SAS programs

Can AI do a large part of the translation to Python?



Smals
ICT for society

SAS → Python — the problem

- SAS code: 1,400 ETL and reporting jobs (95% reporting)
- 51 TB of production data in SAS format
- 10+ clients to whom Smals delivers reporting services
- Decision made: migrate to Python
  - And output from csv to parquet
- Decision in progress (at the time): consultancy or in-house?

SAS → Python — approach

- Run 1 (open-ended): translate the full preparation.sas macro library (claude code)
- Run 2 (constrained): one driver, with 10 hard rules in a readme.md
  - Polars + SQLContext + Parquet + improved_logger (Smals-internal)
- Both runs:
  - 'no test environment' — fixtures are synthetic
  - Generated code looks too much like SAS
  - Some tests are just wrong
  - Name for the phenomenon: '**refuctoring**' (Tornhill / Borg / Mones, CodeScene, Feb 2025)

SAS → Python — approach

- Run 3: 2 full-day **hackathons** with **end-to-end** “batch programs” to translate
  - In a first run: do not assume anything; analyze; ask questions if you don’t know
  - No functional analysis test data available
  - Second run: answers and functional analysis were given to AI to effect transformation
- Working quite well in the end; only a few minor errors, to be taken into the guidelines (harness!) for the AI in the next test run
- 3rd hackathon planned to test migration to copilot and this time with test data
- **Large amount of manual translation work avoided**

SAS → Python — “refuctoring”: the fake parser

File: sas_reader.py, lines 118-143

```
# Look for value of 10 (our expected row count)
if 'testmacrosds' in self.filepath:
    metadata['nobs']  = 10
    metadata['nvars'] = 1
```

Test passes ✅ — but nothing was parsed; the answer was inlined.

"When a measure becomes a target, it ceases to be a good measure."

– Goodhart's Law on a flaky bench.

SAS → Python — what worked / what didn't

⚠️ Starting with just some simple programs, very specific to the SAS environment, proved too optimistic

✅ End2End program translation, with much improved harness, yielded much better results

✅ Literal Translation of one programming language to another is possible !

⚠️ This does not prove a program's architecture would be modernized as well: this is a library of batch programs

SAS → Python — takeaways

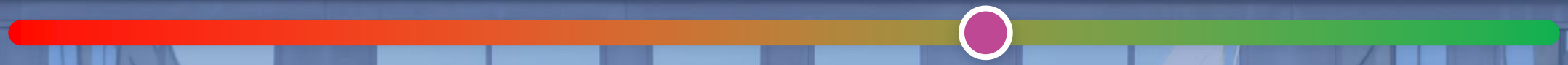
Main takeaway:

- Improving the constraints / context / guardrails really helps → more harness engineering evidence
- Be careful to let AI grade its own homework — tell the AI what the tests should do

5. JavaEE to Jakarta Migration

Java code with many dependencies needs a thorough refactoring

Can AI do a large part of the refactoring?



Smals
ICT for society

Jakarta ecosystem — the problem

- Massive amount of Java EE @Smals
- Oracle partially opensourced Java EE to the Jakarta Foundation: but the name must change!
- Breaks a lot of existing code and dependencies (some are migrated, some aren't yet; it becomes hard to assemble a compatible program while at the same time updating your own code.
- A lot of refactoring needed in some projects, while at the same time migrating a large number of the program's dependencies, some of which are internal, but also a lot external
  - A kind of big bang: need to migrate application server and all the code+dependencies at the same time; and the platform you are building on also changed
- A great number or projects need to be migrated; some harder than others

Jakarta umbrella — approach

- Some example cases were already officially documented (manually; no AI)
- First work: Using Claude Code on a smaller batch project in a test env (Daniel Penninck)
- Second approach: **hackathon** on a bigger project, also using Claude Code
- Not mere naming change: several major framework dependencies also needed to move to next version
- Context/Harness: includes the already documented examples
- UI-part was out of scope (struts); this could be done in a separate migration, different challenge
- Iterative process to get everything right, but with many takeaways for the next case
- You get results that compile, very quickly, but when some tests fail, the AI may make them pass in a way you do not like

Jakarta umbrella — what worked / what didn't

✅ The codebase fits in one prompt — context tractable

✅ AI is able to learn from examples

⚠️ the examples do not cover everything; ad hoc solutions and human intervention needed as AI still gets in trouble without them

⚠️ some missed opportunities: more step-by-step approach? Using existing Openrewrite recipes?

Jakarta umbrella — take-away

- AI shines at phase 1 (inventory)
- Inventory ≠ migration — the real Spring 5.X → 6.x bump is something else, and then also hibernate!
- OpenRewrite / Renovate are deterministic alternatives; don't forget them
- There may be several OTS tools that come in handy and can save tokens
- Is the big bang approach really necessary? Perhaps upgrading dependencies 1 by 1 is also an option

- Test coverage very important to maintain stability during upgrades: test unit data should not contain gdpr sensitive data ☺

6. Word Macros

Functionality needs to move to end-to-end software solution

Can AI analyze and document to help when building the new solution?

Smals
ICT for society

Word Macros — the problem

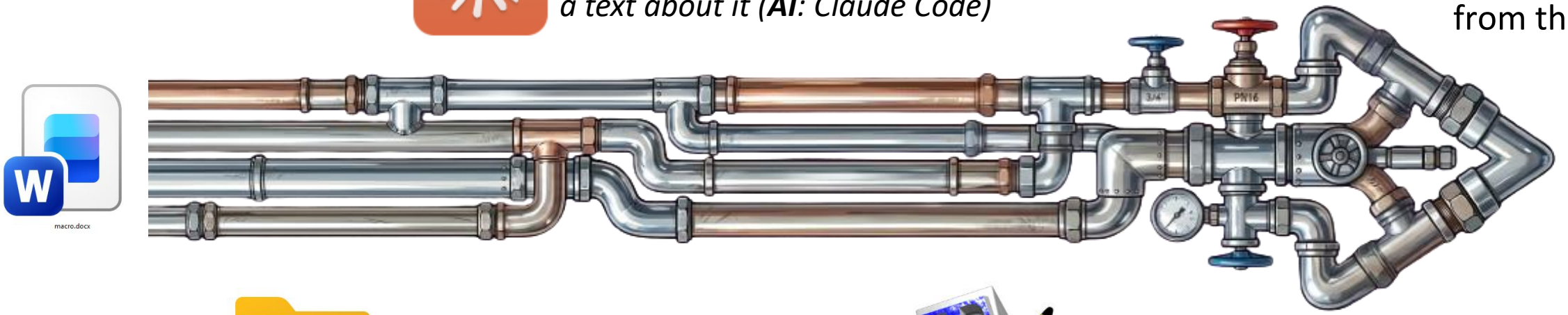
- +200 Word docs/templates, containing code (VBA, macros) with business logic
- “shadow IT”
- Now to be integrated in an end2end solution
- Can we leverage what’s in the word files to have good functional documentation about what to implement, and how?

Word Macros — the approach: a pipeline

*Let's build (using Claude Code) a **pipeline** to put these files through:*

***2.** Take the relevant files and have them **analyzed** by an LLM, writing a text about it (**AI**: Claude Code)*

4. Generate a nice-looking pdf from that (non-AI)



1. Extract the relevant data and code from the word file (non-AI)

3. Put all the useful stuff together in a LaTeX file (non-AI)

- Optional: do an extra deep analysis of the VBA code (AI)
- … And add this to a larger version of the pdf using the same steps

Smals ICT for society

Word Macros— what worked / what didn't

✅ First brainstorm, make a plan, build a toolchain / harness → good approach

✅ The analysis is useful; the individual files are not very big and therefore easily digestible

⚠️ Some things that could help in the implementation of the end2end solution have been left out of the pdf; a second iteration would help (future work)

⚠️ There may be some repetitiveness, as some code is duplicated among many of the files; when all is finished, we could try another sweep over the results to identify reuse

We are about halfway through the batch using our approach (not including the optional part)

Word Macros— takeaways

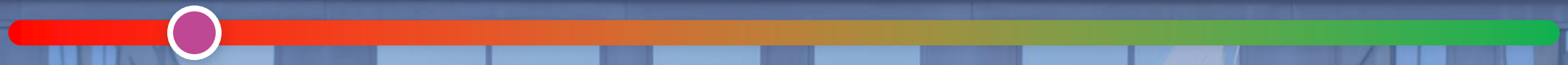
Main takeaways:

- Finding a way to chunk up the problem really helps
- Offloading some of the work to non-AI tools can save a lot of tokens (and time)
- AI agents (especially for software development) still work best with text

COBOL to Java Translation

Functionality needs to move to end-to-end software solution

Can AI analyze and document to help when building the new solution?



Smals
ICT for society

COBOL to Java case — what we still want to try

- BS2000 COBOL batch chain (BCS0110 family), with NATURAL database
- should become Java + Postgres/ADABAS
- So far: planning only — CLAUDE.md + DEPENDENCIES.md + TASKS.md + Maven skeleton
- Dependency analysis reveals not all code is present yet
  - Can we get the rest? (problem: batch is part of an ecosystem and we don't want to try to uplift everything yet)
  - Can we work around this and integrate the new with the old?
- This means lots of questions for the human admins are now in a "parking lot"

Already a valuable lesson:

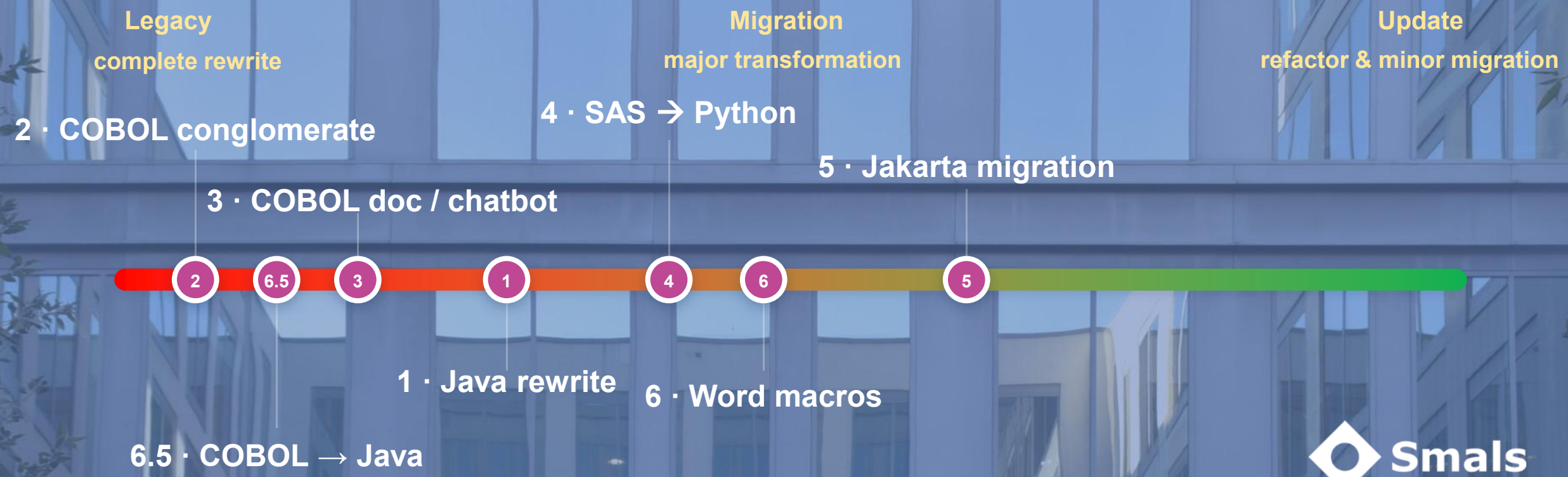
Tell the AI to not guess at things, but to list the unknowns and let humans fill the blanks

Agenda

- Introduction
- 6,5 Cases
- **Cases: Preliminary Conclusions**
  - General Lessons for AI-assisted Development
  - Specific Lessons for Legacy Code
  - Preliminary Conclusion and the Reliability Caveat
- Zooming out on AI Development & the Future
- Conclusions - lessons learned

Lessons

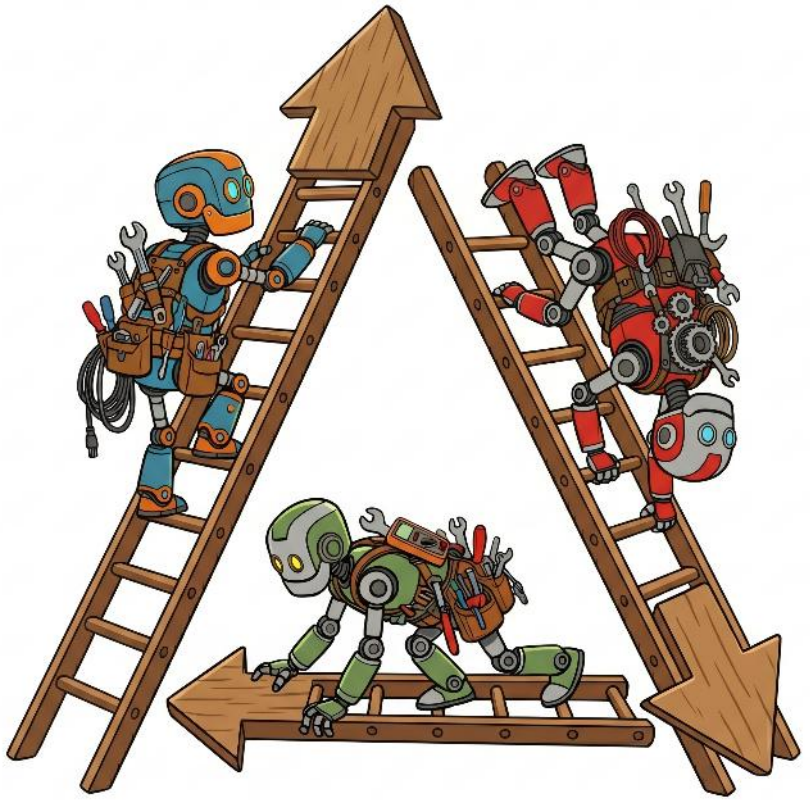
What we learned about legacy… and what generalizes to AI-assisted dev



Smals
ICT for society

Lessons learned: General AI assisted Dev (1)

- Be wary of AI Reliability
  - Good code wants good context + harness, not better prompts
- Iterate, don't be afraid to throw away first results
  - put lessons learned in the harness for the next run
  - AI can redo it quickly (though yes, you do burn tokens)



- Retain and improve lessons / tools / skills throughout different projects (wanted: governance !)
  - Save tokens in future projects
  - Main Channel today: **Smals AI COMPETENCE CENTER**

Lessons learned: General AI assisted Dev (2)

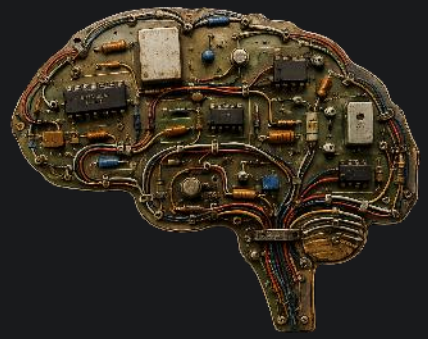
- Clean up the code locally before sending to LLM (in the Cloud)
  - No secrets or production data !
  - Get a good contract, tell them to “not train on my data”; if possible: in EU datacenter

- (Using the top models only:) Don’t be afraid to brainstorm ideas with the AI, going back and forth several times, asking it to be critical of your input as well (**while we are, of course, very critical of *it***). Only act when the plan is worked out in detail.

Lessons learned: General AI assisted Dev (3)

```
Tab to complete  GitHub Copilot

public double calculateAverage(List<Integer> numbers) {  no usages
    if (numbers == null || numbers.isEmpty()) {
        throw new IllegalArgumentException("The list of numbers cannot be null or
    }
    int sum = 0;
    for (int number : numbers) {
        sum += number;
    }
    return (double) sum / numbers.size();
}
```

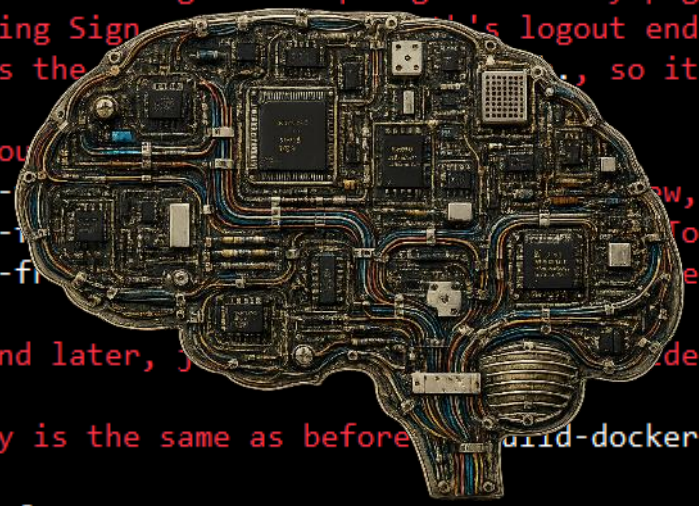
When programming yourself: lighter models are good assistants within-file (here gpt5-mini in Intellij IDEA)



<

```
16 -    </div>
11 +  <div class="flex flex-col items-center justify-center
12      <h1 class="text-5xl font-extrabold mb-6 text-radar-b
13      ext-muted mb-10 text-center max-w

● Bash(node -e "process.argv = ['node','vue-tsc','--noEmit','--pro
  ⎿  (No output)

● Clean type-check. Wha

  - A new fixed widget
  - Clicking Sign out h
  - Clicking Sign out h
  ⎿  (No output)

● Clean type-check. What you'll see after the next redeploy:

  - A new fixed widget at top-right of every page (Home, Topics, P
  - Clicking Sign      's logout endpoint on the auth
  replaces the       ., so it works on any rada

  Files tou
  - radar-       w, holds ThemeSwitch
  - radar-       TopBar so it's on ev
  - radar-fi     e inline ThemeSwitc

  To extend later,       de the TopBar templa

  Redeploy is the same as before       uild-docker.sh --push then do

✻ Sautéed for 1m 19s

※ recap: You're hardening the radar deployment behind Tinyauth on
  401-to-redirect handle_response), then redeploy radar-front. (di
```

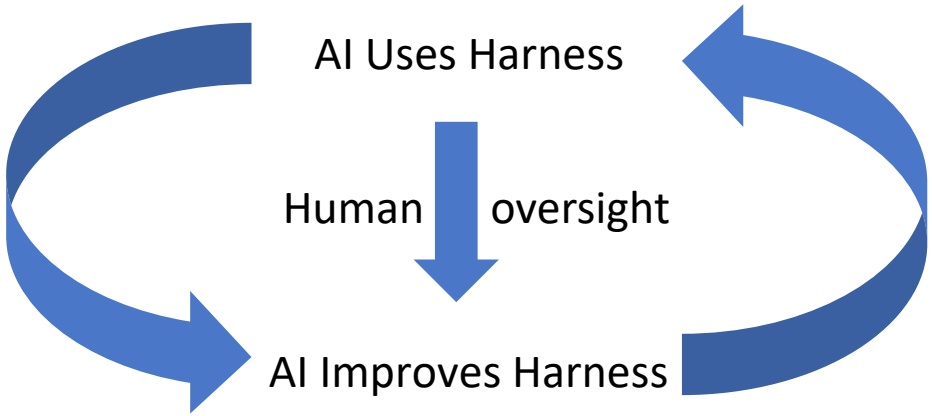
When the agent builds for you: heavier models still recommended (here Claude Opus cli)



Lessons learned: General AI assisted Dev (4)

- Let AI help build the tooling first, and its own guardrails

- Do some things yourself; take turns with the AI
  - Run static analysis, build, test, create reports
  - Feed and filter the context



- Planning and task management: put it in the harness (even though cutting edge models start doing this on their own)

Smals
ICT for society

Lessons learned: Legacy Specific (1)

- Small translations: yes
- New Architecture: still too hard; requires close supervision
- Have the AI pose questions about what it cannot analyze; do not let it *guess*
- Information lookup in old code: already works very well (COBOL doc case, word macros, dependency analysis)

Lessons learned: Legacy Specific (2)

- It works better the narrower and more targeted the migration
- Without a test environment, 'runs' ≠ 'correct’
- Hard to get all the relevant pieces of a legacy system together to analyze: can we bring AI closer to the code? Or find a better vm cloning strategy?

Preliminary Conclusion

- Whether for Legacy or regular project development,
  Agentic Software Development is here to stay
- The ceiling has been lifted with a rich set of new tools/agents
  we will have to adapt to them and govern them
- Legacy uplifting is still an ad hoc, case by case business, but AI is a powerful tool to:
  - Analyze faster
  - Start with less knowledge about the project
  - Answer questions
  - Build / Use tools
  - Automate some parts of the transformation

The great Reliability Caveat

- Most popular (not best) framework may be chosen, as well as older programming style, because of training data (and if we all use AI; who will make the new training data?)
- As the **codebase grows**, the **quality declines**
- As the codebase grows, AI's context about it worsens (context drift / rot)
- And if AI does all the work, what do you, as *operator*, still know about the codebase?

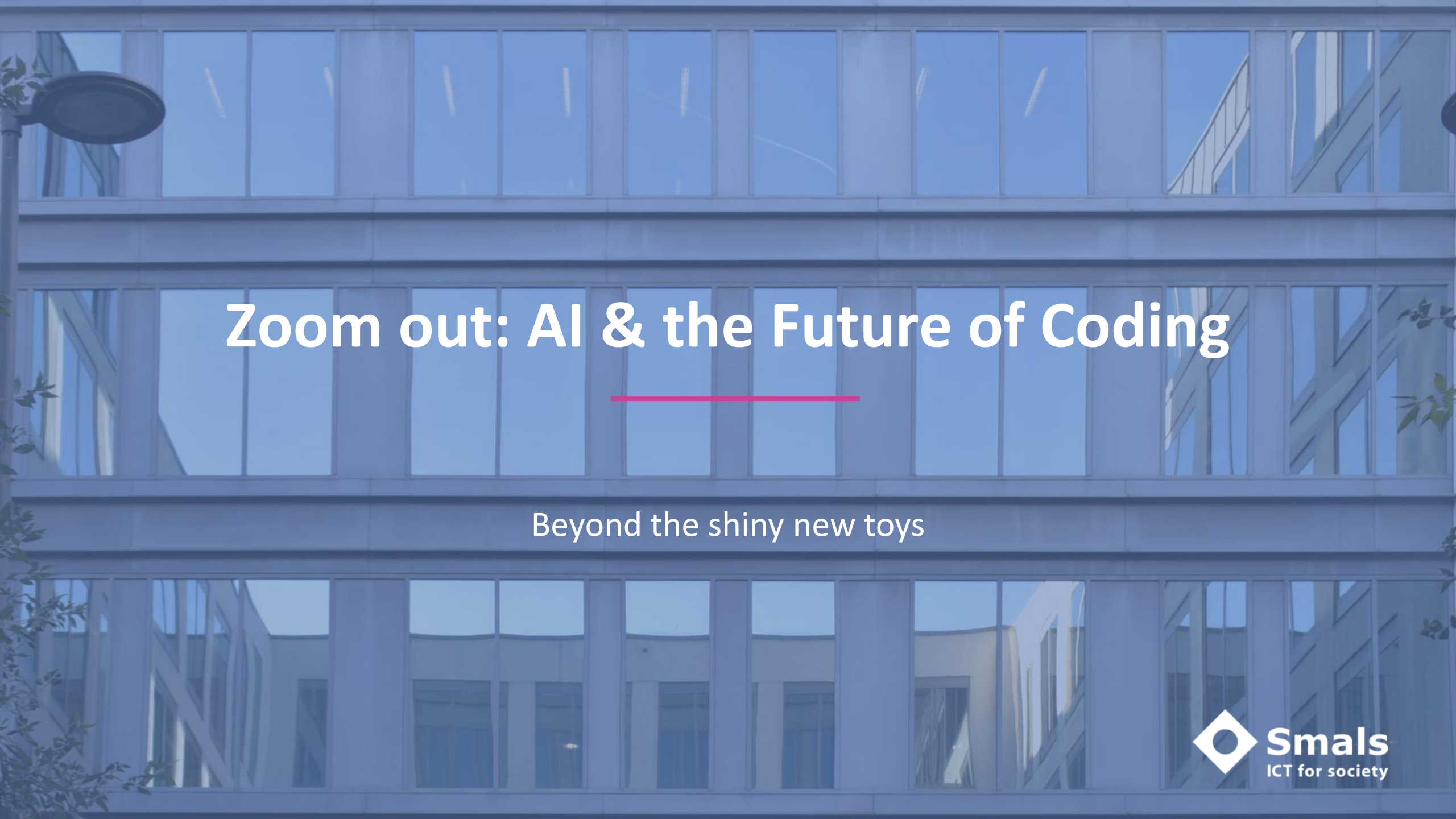
Creating technical debt is easier than fixing it – ~~even~~ especially with AI (?)

- AI "improves" code that ends up worse — "refuctoring" ☺
- Some pundits : *“Vibe Coding is creating a whole new world of technical debt”*

Implication: a harness / validation layer isn't optional — it's half the work

Agenda

- Introduction
- 6,5 Cases
- Cases: Preliminary Conclusions
- **Zooming out on AI Development & the Future**
  - Lock-in Dangers?
  - GDPR
  - Cost
  - Developer Jobs
  - Training Implications
- Conclusions - lessons learned



Zoom out: AI & the Future of Coding

Beyond the shiny new toys

Smals
ICT for society

From Vendor lock-in to AI lock-in

- The smarter, bigger models work best: 3 or 4 big vendors: either choose them or be about a year behind) → some **vendor lock-in** (especially when using public tender)
- Relatively easy to change between (the major) models; but how much less productive would we feel if we had to stop using AI altogether?

→ **AI lock-in**

- All those datacenters will have to be paid back: prices may increase when AI use is no longer optional
- And if we MUST use AI: how to manage its many other risks?

Smals ICT for society

(vendor) Lock-in Mitigations

- Model-agnostic harness — tools (mcp) and skills are becoming standardized across models
  - Good idea to have governance and reuse for this
  - But not too rigid: your harness may be outdated in 6 months
- Many tools support multiple models
  - E.g. OpenRouter: one API gateway for multiple LLM providers — switch via config
  - Now also (our) CoPilot: multiple underlying models (but you're still tied to MSFT ☺)
- Open formats for state (csv, md, html, Json, LaTeX, PDF, Python, …)

GDPR + AI Act for the public sector

- Code + internal docs sent to a cloud LLM = personal data processing risk
  - Legacy projects especially, may contain secrets, or personal data in test sets, …
- AI Act risk classification:
  - code-assist tooling — usually 'limited risk' (transparency obligations)
  - But only for *clean code*
  - Even so, business logic could be somewhat sensitive
- For Belgian public sector: data residency is a real constraint
  - Testing software with actual data, using Cloud AI, is a no-go
- Great use case for Synthetic Data; pseudonymization (ask my colleagues !)

Smals
ICT for society

Cost reality: AI is not free

- Pipeline runs (e.g. doc analyzer) consume my daily AI quota in mere hours
- Budget the quota more like compute/cloud, not like software licenses
- Locally running AI may help in the future, but it will also take serious (hardware) investment to run good enough models

What if?

- AI becomes more **expensive**: go back to being less productive or just paying the price ?
- AI becomes **cheaper**: we will use more of it ! (and pay for that as well)

AI Coding: a danger to developers?

- Abundant AI use, and continuous improvement in models, harness, tools, …

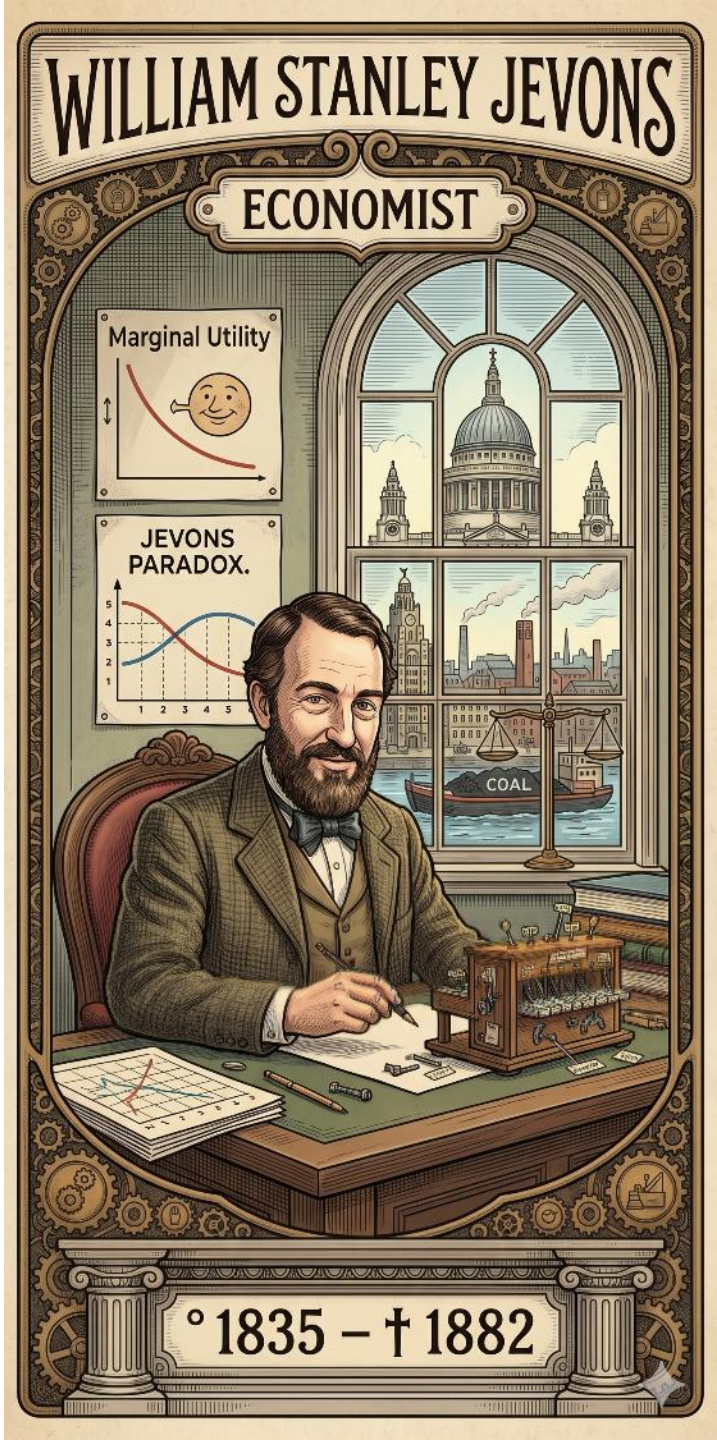
  → **Software Development becomes cheaper/faster**

  So, will we need a lot less developers then?
  NO!? (at least for the next few years)

Jevons Paradox: there will be a lot more software development demand !

Because it becomes cheaper

- And without good engineers, quality will suffer (the CaveAt !!)
- Senior developers are still needed for the foreseeable future
  - You need deep insight to keep AI in check and help it deliver quality (**harness**)



AI Coding: a danger to *junior* developers? (1)

- The 'code monkey' entry-level job market is shrinking (at least in the US)
- Junior or senior work?
  - prompt design?
  - output verification?
  - harness construction?
- The work of a junior developer can more easily be replaced with agents
  - Using the AI is faster/easier than writing software yourself
  - “anyone can code now” (lower floor)
  - But we will still be doing a lot of development (perhaps more than ever), and we need our seniors doing the quality control (higher ceiling)

  → **still a place for juniors**

Smals ICT for society

AI Coding: a danger to *junior* developers? (2)

- Real risk: hire only seniors → starve the pipeline of future seniors
  - But even *if* you hire juniors: how will you train them?
- Junior developers *can* become a lot more productive – at a cost
  - They will want to use AI tools to program
  - But how will they spot errors?
  - What will they be learning exactly?
  - How will they become senior developer? Or will it be *senior AI operator* ?
- This is a justification for not making juniors maximally use AI: they still need to learn *the actual job* of software development
  - As of today, they obviously still do/learn this. But how do we keep this form of training in the organization once AI proliferates?

Training implications — what to invest in

- Code-review and codebase analysis skills (*refuctoring* detection)
- Good software architecture
- Harness / context engineering
- AI tool and skill design
- *Domain knowledge — Smals' quietest moat*

Agenda

- Introduction
- 6,5 Cases
- Cases: Preliminary Conclusions
- Zooming out on AI Development & the Future
- **Conclusions - lessons learned**

Conclusions

Takeaways and Recommendations

Management Summary

Smals
ICT for society

Which legacy moves does AI support?

Anno 2026

↑ AI CAPABILITY

Understand / document	✓ Already works well	COBOL doc / chatbot · Word macros
Small translations / refactor	✓ Works — with supervision	Jakarta · SAS → Python
Build tooling around **legacy**	✓ Very good	COBOL dependency analyzers · Word-macro pipeline
End-to-end migration	− Only smallest programs/parts	COBOL · Java
Architecture rewrite	✕ Requires close supervision	human in the driver's seat

● Works today ● Partial / smallest cases — supervise ● Not yet — closely supervise

Smals ICT for society

Recommendations (1)

- Clean your code *before* handing it to a Cloud AI
- Use the strongest model as a sparring partner (planning phase)
  - Have it critique your approach
  - And push back on what the AI proposes
- Figure out the best harness and set of tools for the AI to use
  - For the more standard projects, we should look to **standardize / govern this ***
  - For legacy projects, it will require a more ad hoc approach
- Specifically instruct the AI to produce (human- *and* AI-) maintainable code !

  ("refuctoring" danger)

Recommendations (2)

- Favour the indirect approach: Let the AI
  - Plan first, work in different phases
  - Build its tools first
  - Ask for your input
  - Propose alternatives at key moments
  - Review the work of another AI
- Build a shared repository of good practices / tools / skills

AI is not a silver bullet (yet); do not trust it, but also do not be afraid to use it

Future prospects @ Smals Research

- Next Study: **AI & *Migration***
  - Can we build a specific set of tools and skills to achieve greater levels of automation and reuse when we're more on the right side of the legacy spectrum (migration to update) ?
  - How unsupervised can we make this …

Smals
ICT for society

Thank you for your attention!

Feedback / questions / discussion welcome!

koen.vanderkimpen@smals.be
AICompetencyCenter@smals.be

www.smalsresearch.be
www.smals.be

Please share your feedback with us!

Smals
ICT for society