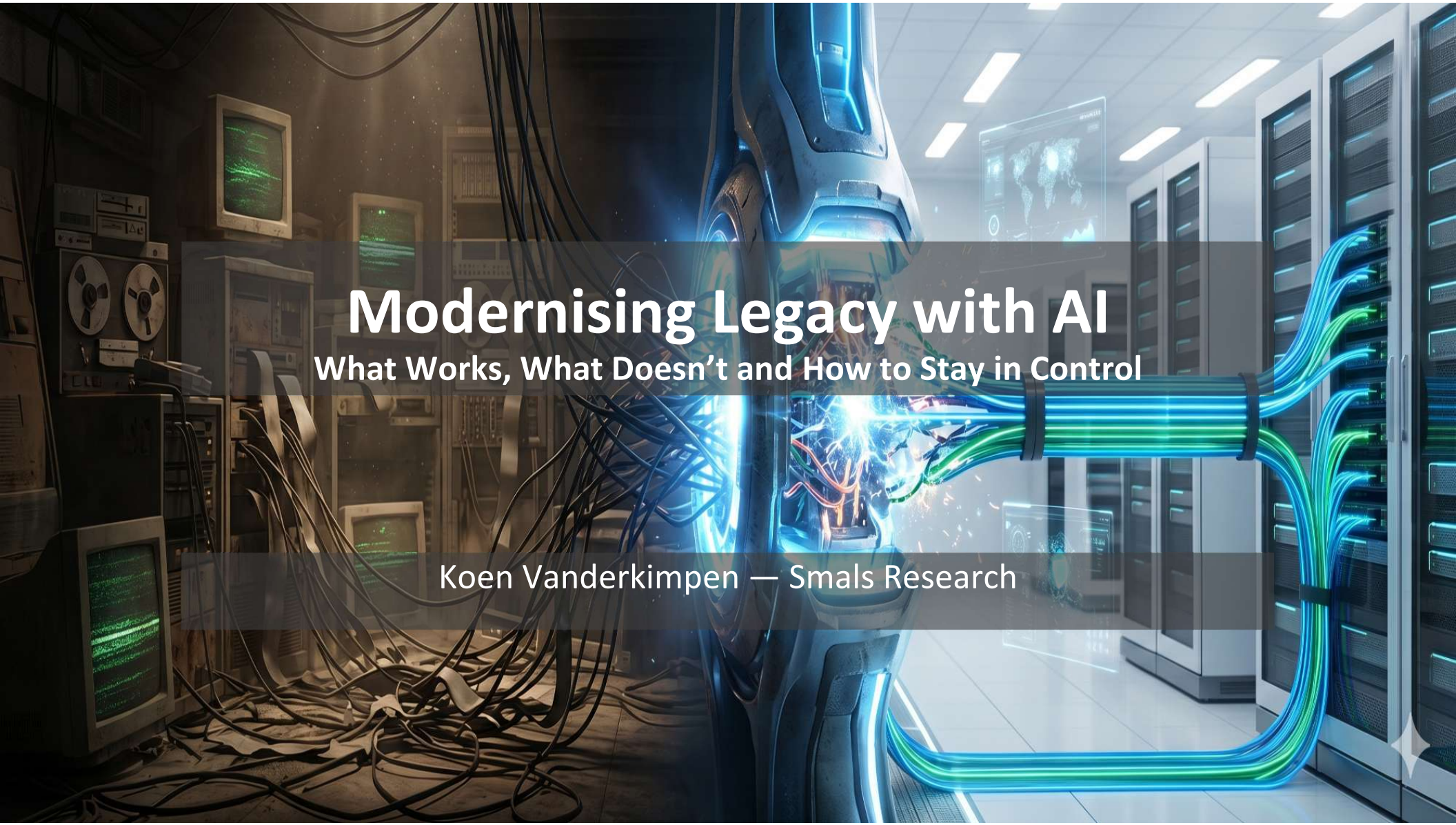


Modernising Legacy with AI

What Works, What Doesn't and How to Stay in Control

Koen Vanderkimpen — Smals Research



Bio



Koen Vanderkimpen

Smals Research, AI CC

Smals
ICT for society



About Smals – Who we are

- Association of >300 public institutions in Belgium
- Subject to public procurement regulations
- “ICT for society” = Strictly focused on eGovernment
- Shared services & cost sharing (non-profit)
  - Application development
  - Infrastructure & Services
  - Staffing
- Only serving member institutions
- >2.000 employees
- >500 million EUR annual turnover
- All expenses are charged back to our members without margin

Smals
ICT for society

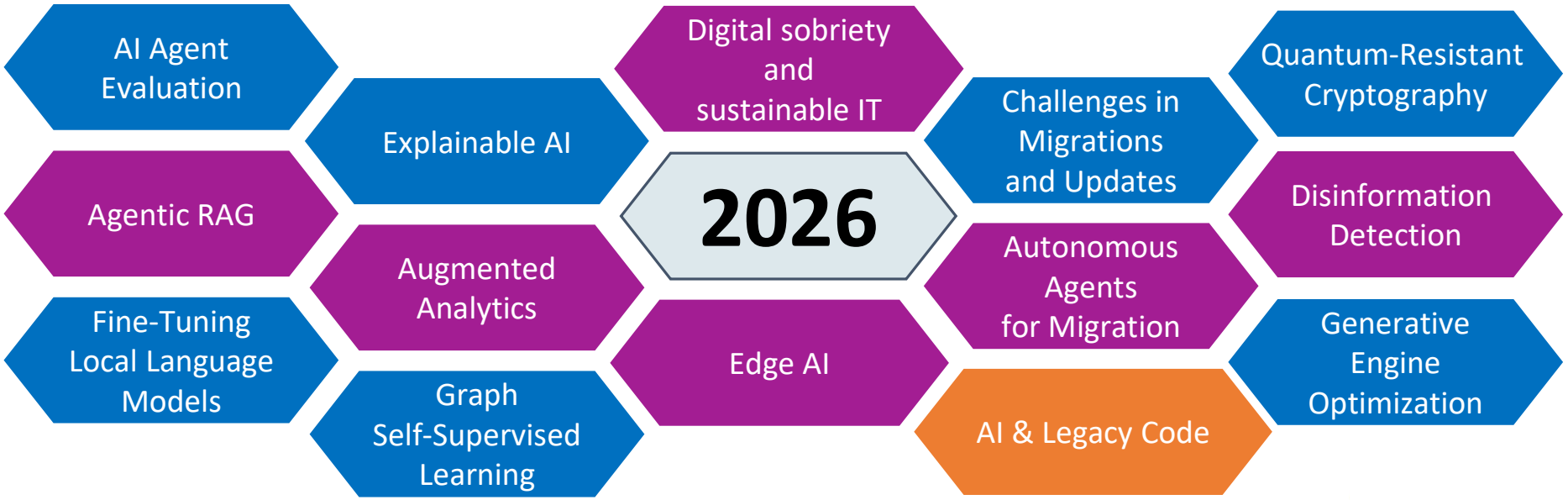
Smals Research 2026

Innovation with new technologies

Consultancy & expertise

Internal & external knowledge transfer

Support for going live



www.smalsresearch.be



Agenda

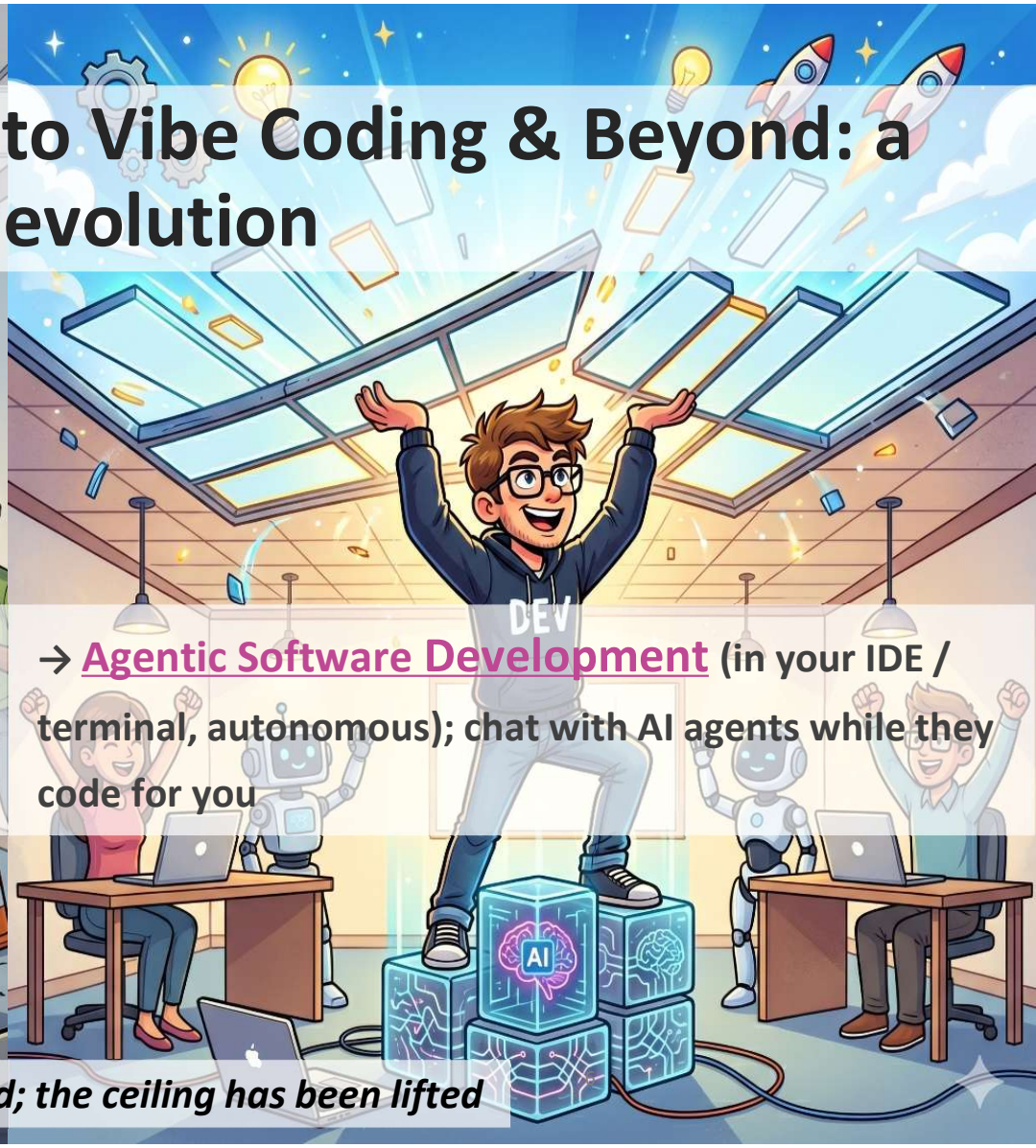
- **Introduction**
  - AI & Development: our take
    - Harness Engineering
  - The Legacy Spectrum
  - AI for Legacy code, specifically
- Cases & Preliminary Conclusions
- Zooming out on AI Development & the Future
- Conclusions - lessons learned

Smals
ICT for society

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



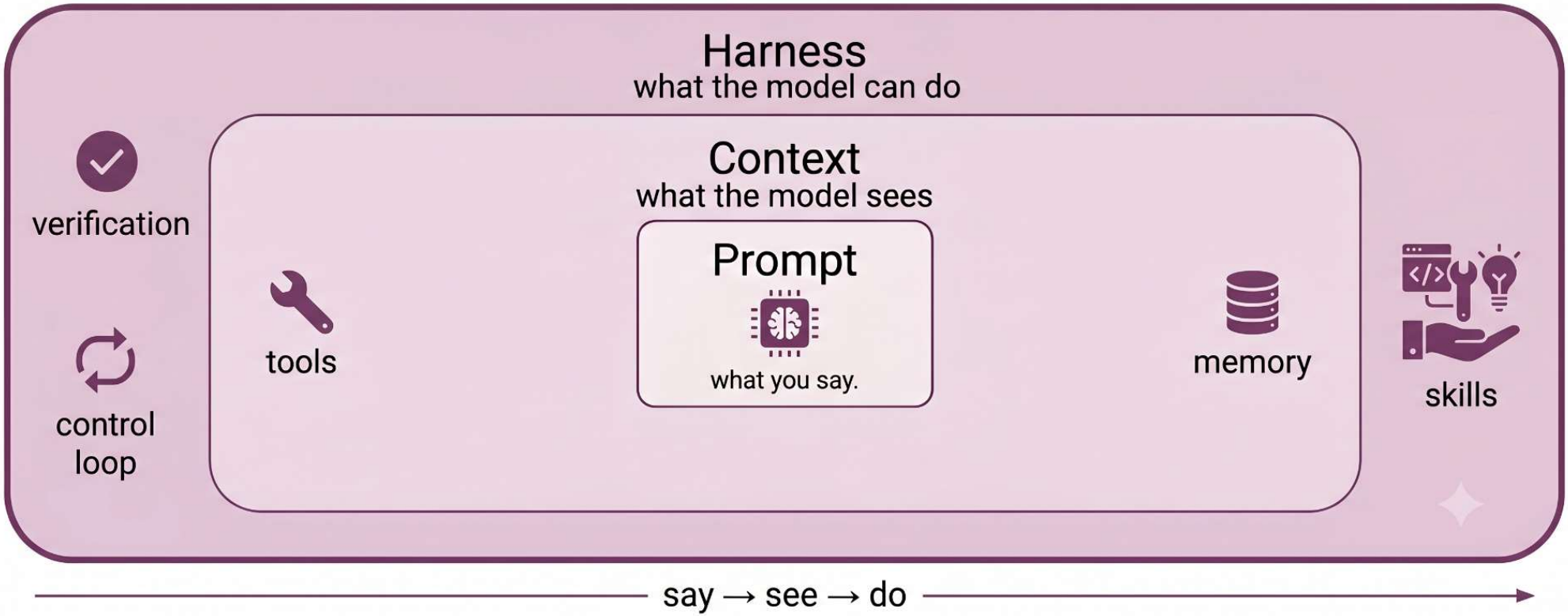
→ Vibe coding (talking to a system that writes an application for you; no skill required)



→ 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

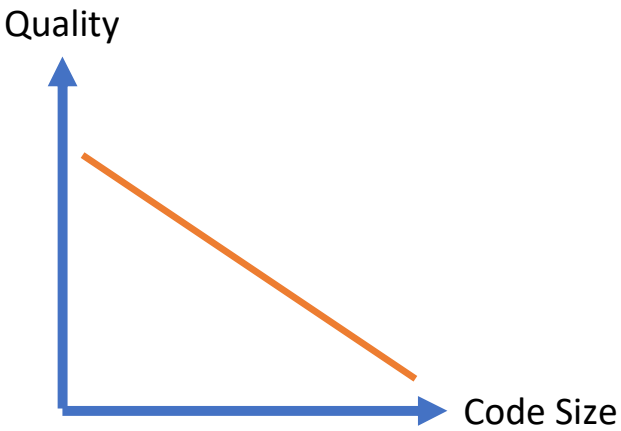
Agentic Coding: towards Harness Engineering



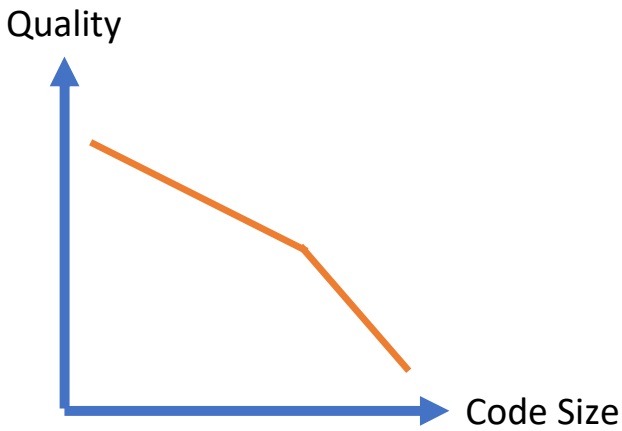
Caveat: Anno 2026 !

Smals
ICT for society

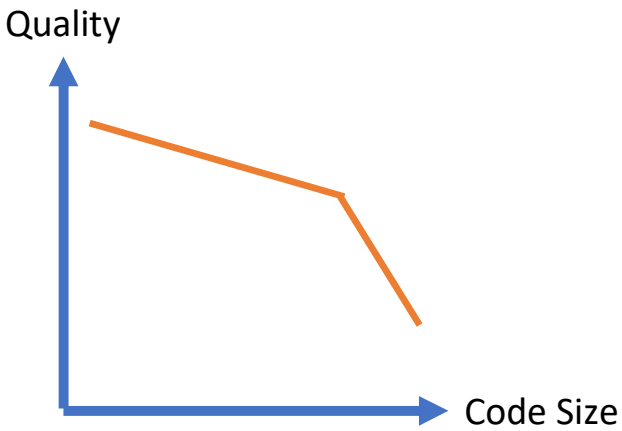
Harness Engineering vs previous (our take)



Prompt Engineering



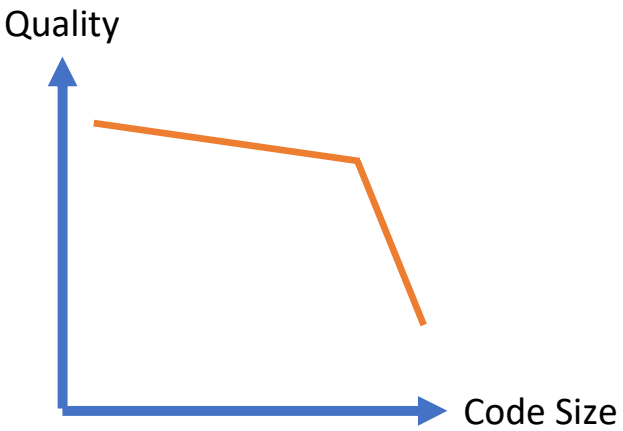
Context Engineering



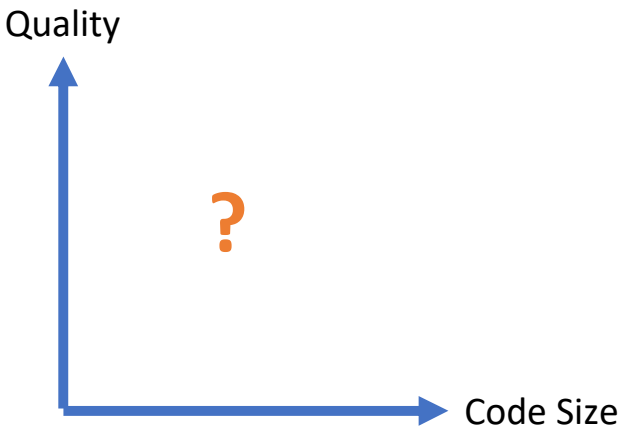
Harness Engineering

Smals
ICT for society

Where can we take this?



Human with experience
in the codebase,
Using AI?



Spec-Driven
Development ?
Loop Engineering

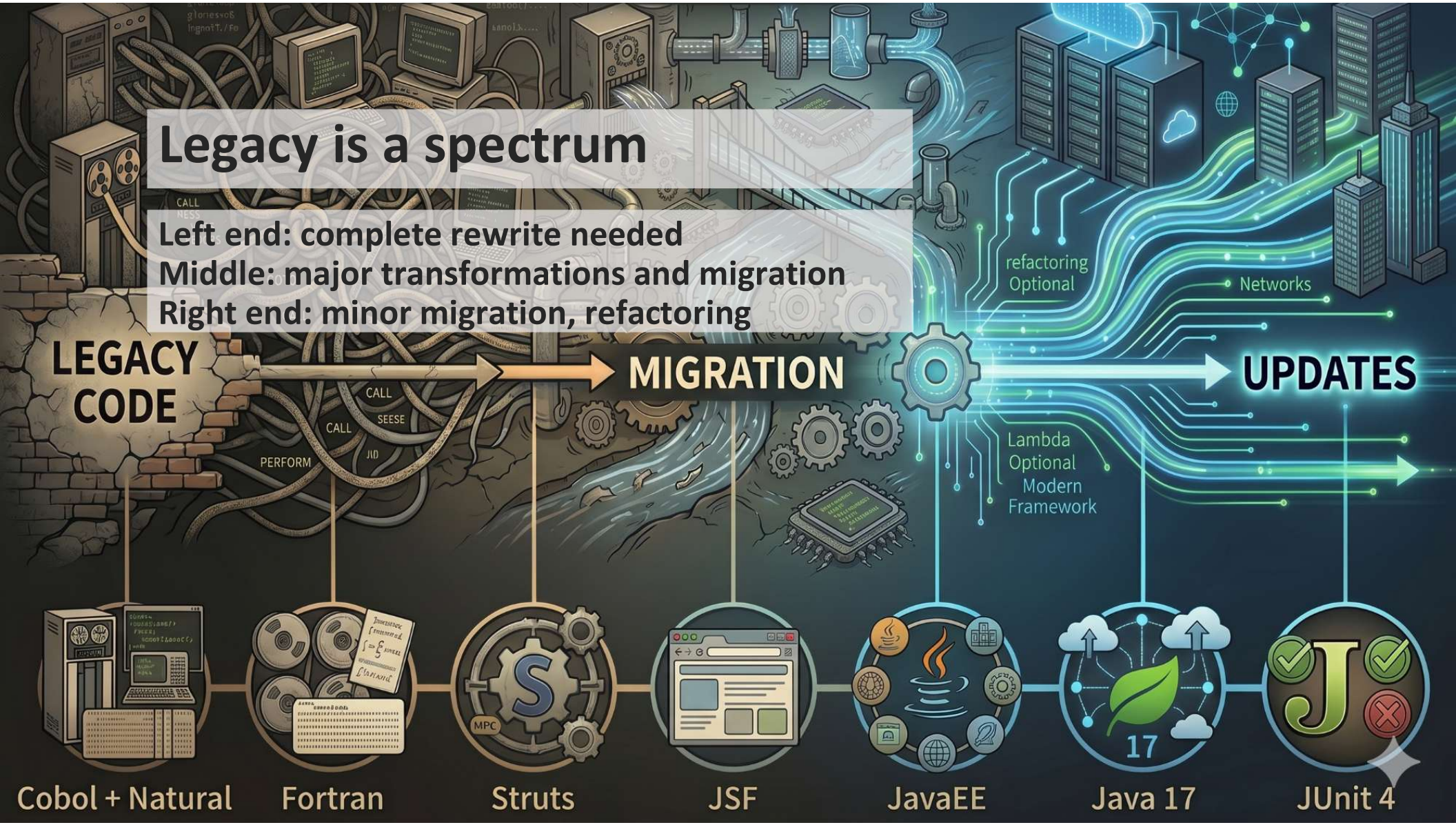
Smals
ICT for society

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
LEGACY CODE
MIGRATION
UPDATES
refactoring Optional
Networks
Lambda Optional
Modern Framework
CALL
SEESE
CALL
JID
PERFORM
MPC
17
Cobol + Natural
Fortran
Struts
JSF
JavaEE
Java 17
JUnit 4

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 (and even then…)
  - 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)
  - Needs explicit guidance (out of the box AI advice sounds like a consultant who looked at your code for 5 minutes)
- Oldest Legacy: system it is on does not support AI…

Smals ICT for society

AI & Legacy Code ≠ AI & Coding

Different pathway to handle legacy: **analyzing / documenting** your old code

- With Legacy, it is sometimes better to get the useful stuff ***out*** of it
- E.g. business knowledge / logic / requirements
- Rebuild something new with that knowledge!
- AI can be useful as “archaeologist”
- Or as “all-knowing” chatbot

AI & Legacy Code ⊆ AI & Coding

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



Smals
ICT for society



Agenda

- Introduction
- **Cases & Preliminary Conclusions**
  - Short Summary of COBOL cases
  - Short Summary of other cases
  - JavaEE → Jakarta: then & now
  - Lessons & Preliminary Conclusions
- Zooming out on AI Development & the Future
- Conclusions - lessons learned

Smals
ICT for society

The Spectrum, and our cases

Legacy
complete rewrite

Migration
major transformation

Update
refactor & minor migration



Smals
ICT for society

COBOL cases — at a glance

Legacy complete rewrite — **Migration** major transformation — refactor & minor migration **Update**



Jakarta — coming up

2 COBOL Conglomerate

Social security · 789 COBOL programs, 216 JCL · obsolete, unsupported stack

Approach: AI-written analyzers (22 Python tools) → dependency graphs; GnuCOBOL upgrade POC

✅ Graphs gave real insight; small programs compile and run on GnuCOBOL

⚠️ No test server, so "runs" ≠ "correct"; generic AI analysis stayed bland

If AI can't handle the codebase size, the tools it writes can.

3 COBOL Docs → Chatbot

Social security · 250+ COBOL programs · no functional docs · key expert retired

Approach: Copilot agent (RAG) on the expert's explainer videos + AI transcripts, in Teams

✅ Published to a defined user group; good first test results

⚠️ Written docs, dependencies and the code itself not used yet

A RAG chatbot can beat documentation: the right answer, in context.

7 COBOL → Java

BS2000 COBOL batch chain + NATURAL DB → Java + Postgres/ADABAS · ongoing

Approach: planning only so far — CLAUDE.md, DEPENDENCIES.md, TASKS.md, Maven skeleton

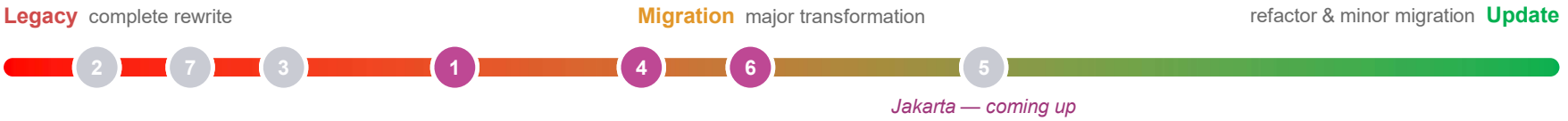
✅ Dependency analysis showed early on that code was missing

⚠️ Batch is tangled in a larger ecosystem; open questions parked for admins

Don't let the AI guess: have it list the unknowns for humans to fill in.

Smals ICT for society

Java & other cases — at a glance



1 Java Technical Rewrite

eHealth · Java 8 + Struts · isofunctional rewrite already decided (2024)

Approach: agentic coding builds a feature in the new project from logic in the old code

✅ Found the right old code (incl. DB access); feature done in a few hours

⚠️ Small feature (<200 LOC), yet architecture needed hand-holding: ~80–90% correct

AI needs rigorous constraints: a good harness.

4 SAS → Python

1,400 ETL/reporting jobs · 51 TB of SAS data · 10+ client organisations

Approach: open-ended vs. constrained runs, then hackathons on end-to-end batch programs

✅ Stronger harness → working translations, few errors; much manual work saved

⚠️ Synthetic tests only; “refuctoring”: AI faked a parser to pass a test

More harness, better results. Don’t let AI grade its own homework.

6 Word Macros

200+ Word templates with VBA business logic (“shadow IT”) → end-to-end solution

Approach: Claude Code built a pipeline: extract → LLM analysis → LaTeX → PDF

✅ Useful functional docs; small files are easy for the AI to digest

⚠️ Some implementation details missed; code duplicated across files

Chunk the problem; offload to non-AI tools to save tokens and time.

Smals ICT for society

JavaEE → Jakarta migration — then & now

Legacy complete rewrite — **Migration** major transformation — refactor & minor migration **Update**



Case lead: Wim Lambrecht, Smals AI CC

5 The challenge

Java EE became Jakarta EE: renaming javax → jakarta breaks servers, frameworks, libraries

Case: social security application suite on JBoss EAP 7.4 → 8, Spring 5.3, Hibernate 5.6, EJB, SOAP/REST, JMS

✅ Company knowledge base with worked examples: the AI learns from them

⚠️ A big bang: server, frameworks and own code have to move together

Ground the AI in your own knowledge and examples.

THEN Guided prompting

Prepare first: a sane codebase (test coverage, no sensitive data) and the team on board from day 1

Approach: one main prompt makes the AI write a plan, phases and a dependency inventory; a human runs it phase by phase

✅ progress.md keeps an audit trail; human in the driver's seat

⚠️ Unknown dependencies need human input and separate "sidetrack" cycles

Start from a sane codebase; involve the product team from day 1.

NOW Beyond prompting

Same goal, more structure: from one long prompt to reusable building blocks

Agentic: specialised agents with their own roles, coordinated by a plan

Spec-driven: a SPEC.md with quality gates (definition of done) per phase

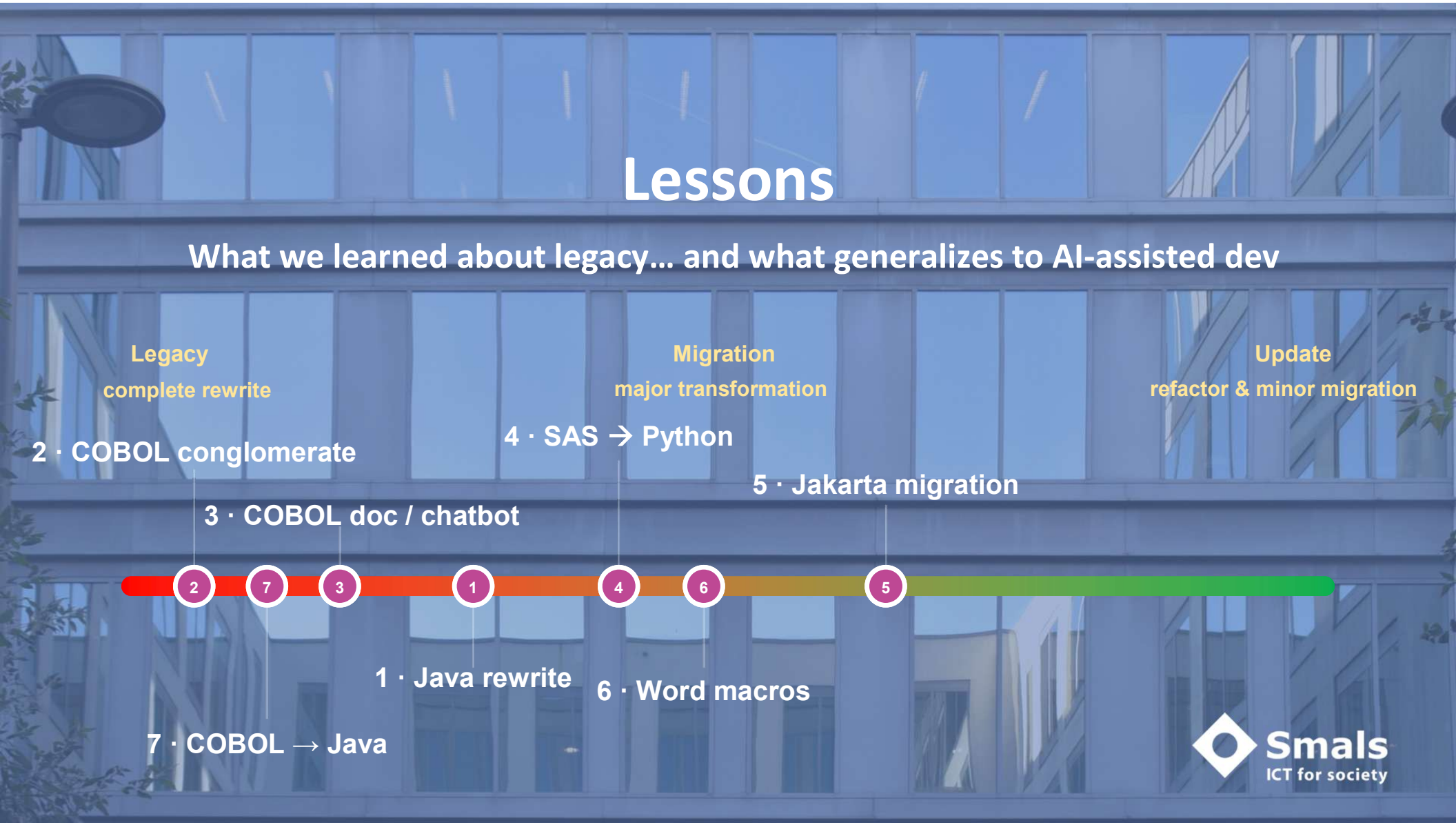
Skills only: one reusable migration skill (EAP 8, Spring Boot, …)

Profitable? Absolutely. Best approach? It depends: be pragmatic.

Smals ICT for society

Lessons

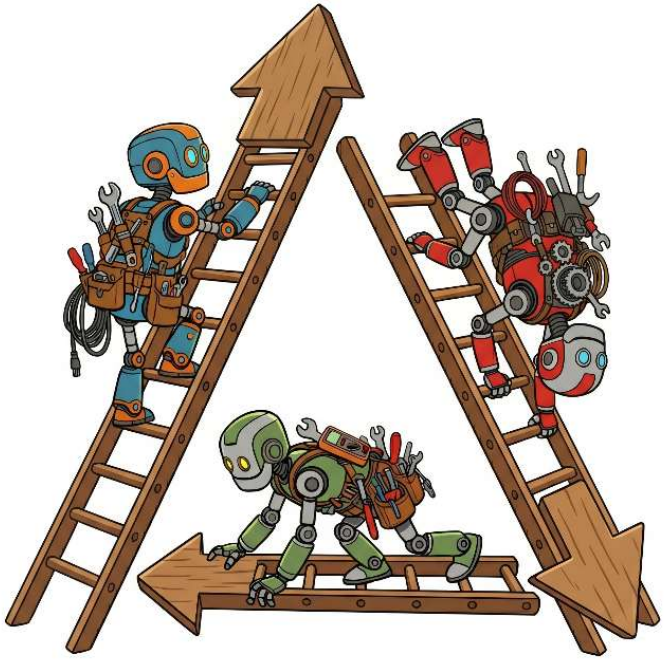
What we learned about legacy… and what generalizes to AI-assisted dev



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)



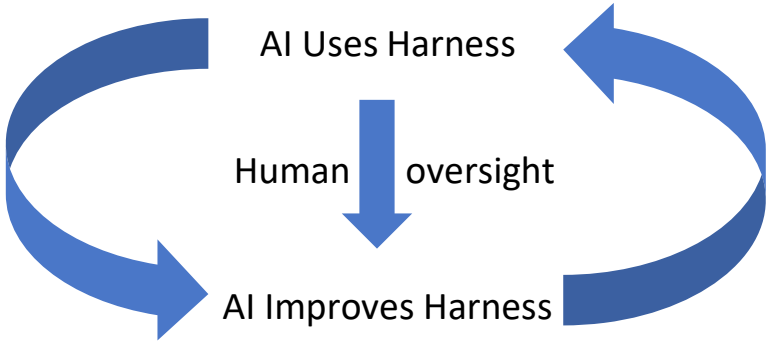
- Retain and improve lessons / tools / skills throughout different projects (wanted: governance !)
  - Save tokens in future projects
  - Big Enterprise? Get a **AI COMPETENCE CENTER** ☺

Smals ICT for society

Lessons learned: General AI assisted Dev (2)

- 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 (until cutting edge models start doing things better on their own)

Smals
ICT for society

Lessons learned: General AI assisted Dev (3)

- 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?

Smals
ICT for society

Preliminary Conclusion: legacy

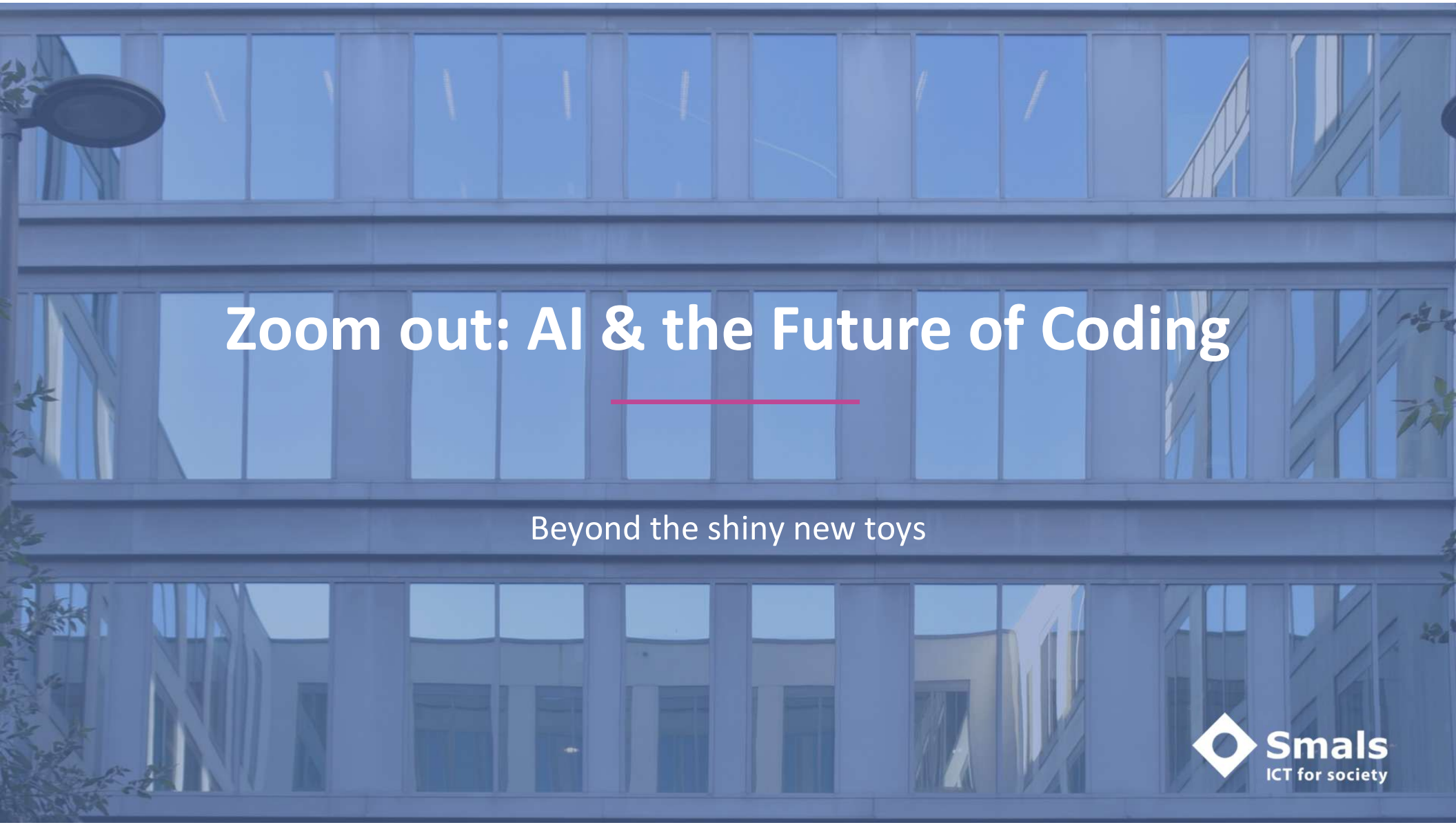
- Legacy uplifting is still ad hoc, case by case, but AI helps you analyse faster, start with less knowledge of the project and get answers
- Narrow and targeted works best: small translations yes, new architecture only under close supervision
- Let the AI list what it cannot analyse; don’t let it guess
- Without a test environment, “runs” ≠ “correct”
- The biggest obstacles are organisational: getting all pieces of a legacy system together for the AI takes company strategy



Agenda

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

Smals
ICT for society


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 6 months to 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

AI Coding: a danger to developers?

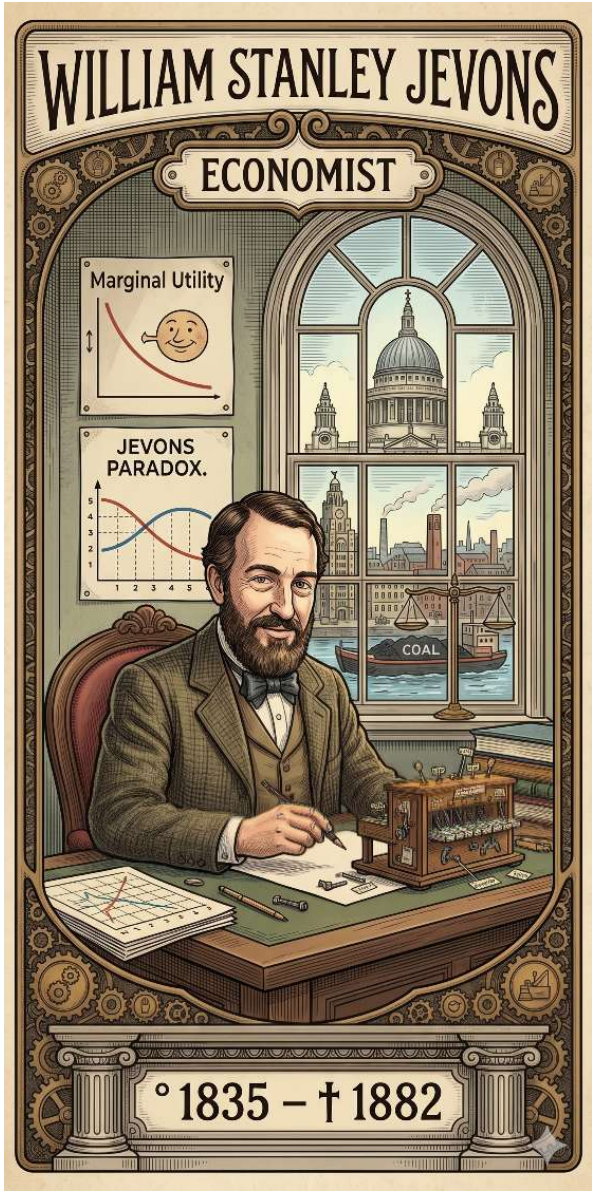
- Abundant AI use, and continuous improvement in models, harness, tools, …

  → **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**)



Training implications — what to invest in

- Code-review and codebase analysis skills (*refuctoring* detection)
- Good software architecture
- Harness / context engineering; Requirements “Driving”
- AI tool and skill design
- *Domain knowledge*

Basically, being critical ☺

Smals
ICT for society



Agenda

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

Smals
ICT for society



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
  - No secrets or production data
  - A contract that excludes training on your data; EU datacenter if possible
- Use the strongest model as a sparring partner (planning phase)
  - Have it critique your approach, and push back on what it proposes
  - Only act when the plan is worked out in detail
- Figure out the best harness and set of tools for the AI to use
  - Standard projects: **standardise and govern this**; legacy projects: more ad hoc
- Specifically instruct the AI to produce (human- *and* AI-) maintainable code!
  ("refuctoring" danger)

Smals
ICT for society

Recommendations (2)

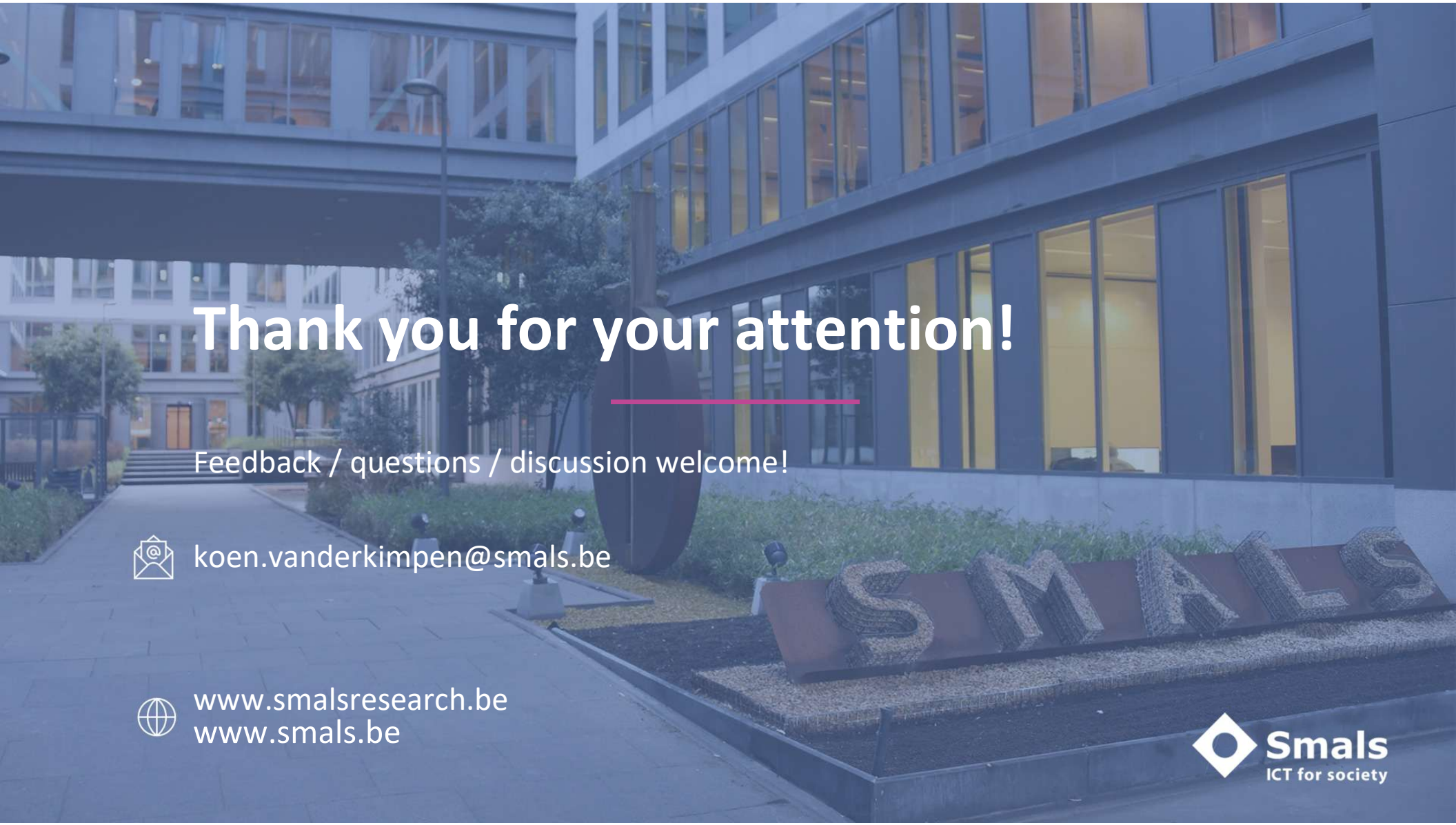
- Favour the indirect approach: Let the AI
  - Plan first, work in different phases
  - Build its tools first
  - Ask for your input
  - Propose alternatives at key moments
  - Review the work of another AI
- Build a shared repository of good practices / tools / skills

Smals
ICT for society

Recommendations (3): after the migration

- A human owns the result: someone must still understand the architecture
  - Review AI-written code (using another AI ?) as if a new colleague wrote it
- Tests are the contract: build them first, on a sane codebase with synthetic data
  - They make every later change safe, whether a human or an AI makes it
- Keep the knowledge next to the code
  - Analyses, dependency graphs, docs, the chatbot's knowledge base
- Version the harness with the code
  - The rules, examples and skills that built it are what you need to change it

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



Thank you for your attention!

Feedback / questions / discussion welcome!

koen.vanderkimpen@smals.be

www.smalsresearch.be
www.smals.be

Smals
ICT for society