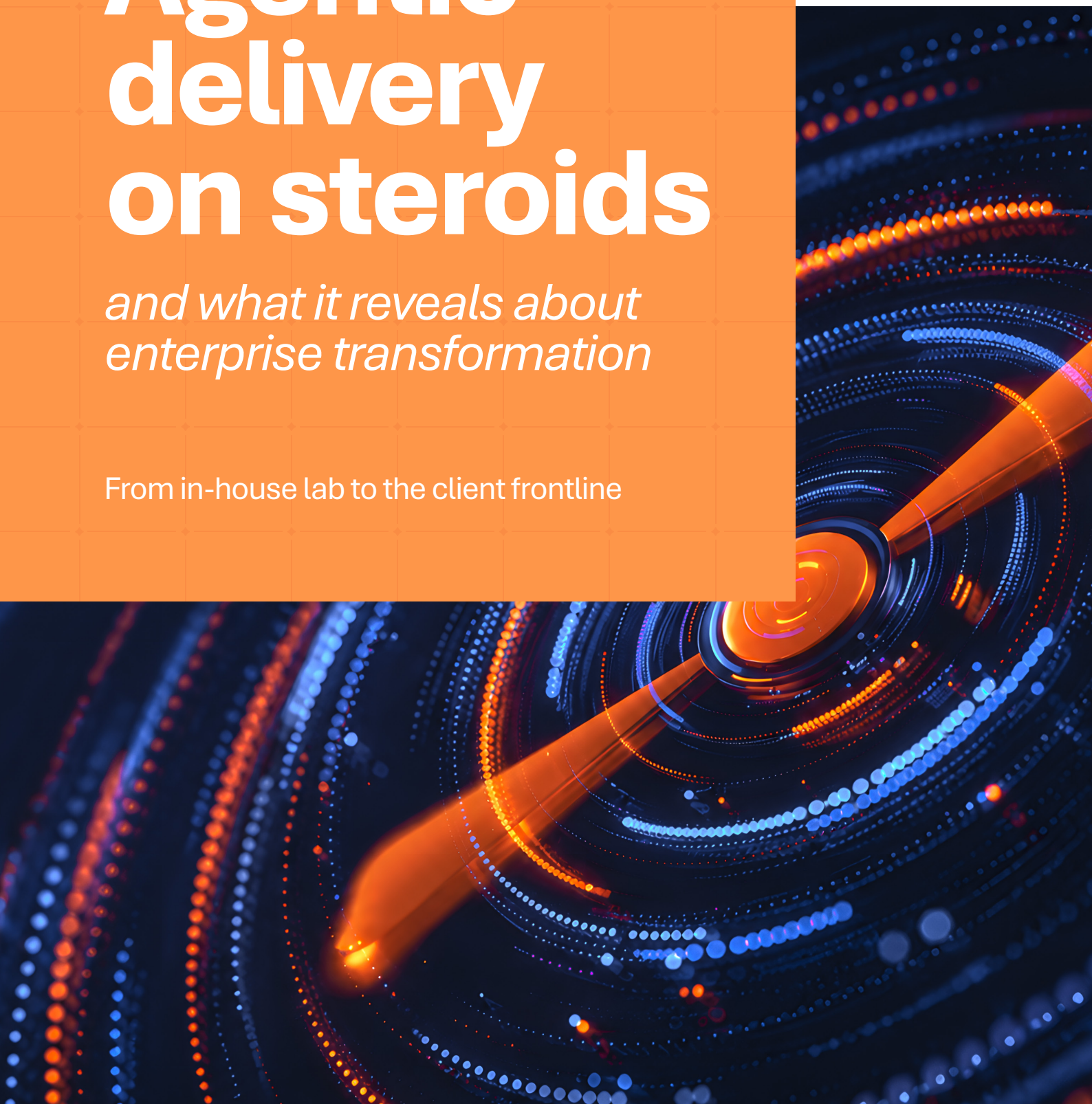
Nova Ventura
MAP. MARK. MOVE.

Agentic delivery on steroids

and what it reveals about enterprise transformation

From in-house lab to the client frontline



THE AUTHORS



Benjamin Geens
Director Digital Transformation & Strategy
Nova Ventura



Barbara Arnst
VP Transformation
Telenet group

June 2026 \ Version 1

Contents

01 Executive summary 4

02 Who this whitepaper is for 8

03 Introduction: why we ran this experiment 10

04 PART 1: Building ACE Ventura - product decisions and learnings 12

4.1 The original use case and how it evolved 13

4.2 Why we built a custom platform instead of relying on out-of-the-box AI 15

4.3 Keeping the agent definition layer in-house 16

4.4 Trust and quality: why we curate what the AI can use 17

4.5 Confidentiality by design: consultants only see what they should 18

4.6 Multi-agent architecture: one agent per service offering 19

4.7 Model routing, cost control, and operational visibility 20

4.8 Making the platform self-improving: learning from usage 22

4.9 Source links and feedback loops 23

05 PART 2: Agentic coding: what we learned 26

5.1 Team setup: exploration helps, convergence matters 27

5.2 Merge pressure: integration pain shows up sooner 28

5.3 The agent memory problem: context drops between sessions 31

5.4 The ecosystem moves too fast: balancing novelty and delivery 32

5.5 Feature creep amplified: building becomes addictive 33

5.6 Quality: standards must be written into the system 34

06 PART 3: A look into the future - how agentic systems will reshape organizations 38

6.1 From IDE copilots to agentic systems across the lifecycle 40

6.2 The impact on operating models: roles, responsibilities, and decision flows 44

6.3 Change management belongs in the operating model discussion 47

07 PART 4: What we are learning at enterprise scale, together with Telenet 48

7.1 Three discovery tracks we are working on together 50

7.2 What we are learning together so far 52

7.3 Perspective of Barbara Arnst from Telenet 54

08 A final word 60

09 About Nova Ventura and Nova Reperta 62

10 Appendix: 64

10.1 Building Blocks for AI transformation: A practitioner reference 64

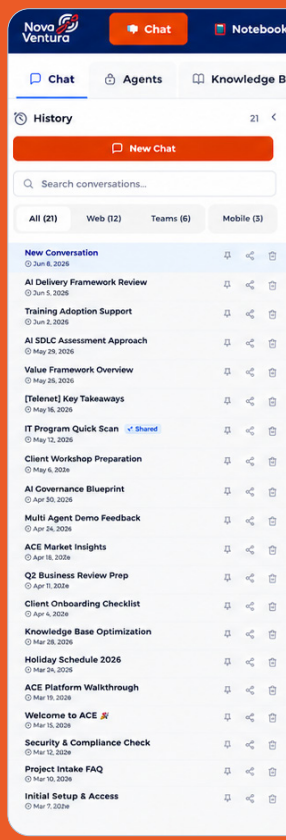
10.2 About the authors 66

1

executive summary

AI is now everywhere, and it is genuinely powerful. Yet for most organizations the hard part is no longer access to the technology; it is turning that access into real value. That gap is rarely technical. It comes down to how knowledge, work, and decisions are organized. So rather than advise clients from the sidelines, we decided to run the experiment on ourselves first, on our own work, with our own people and our own stakes.

That experiment became ACE Ventura.



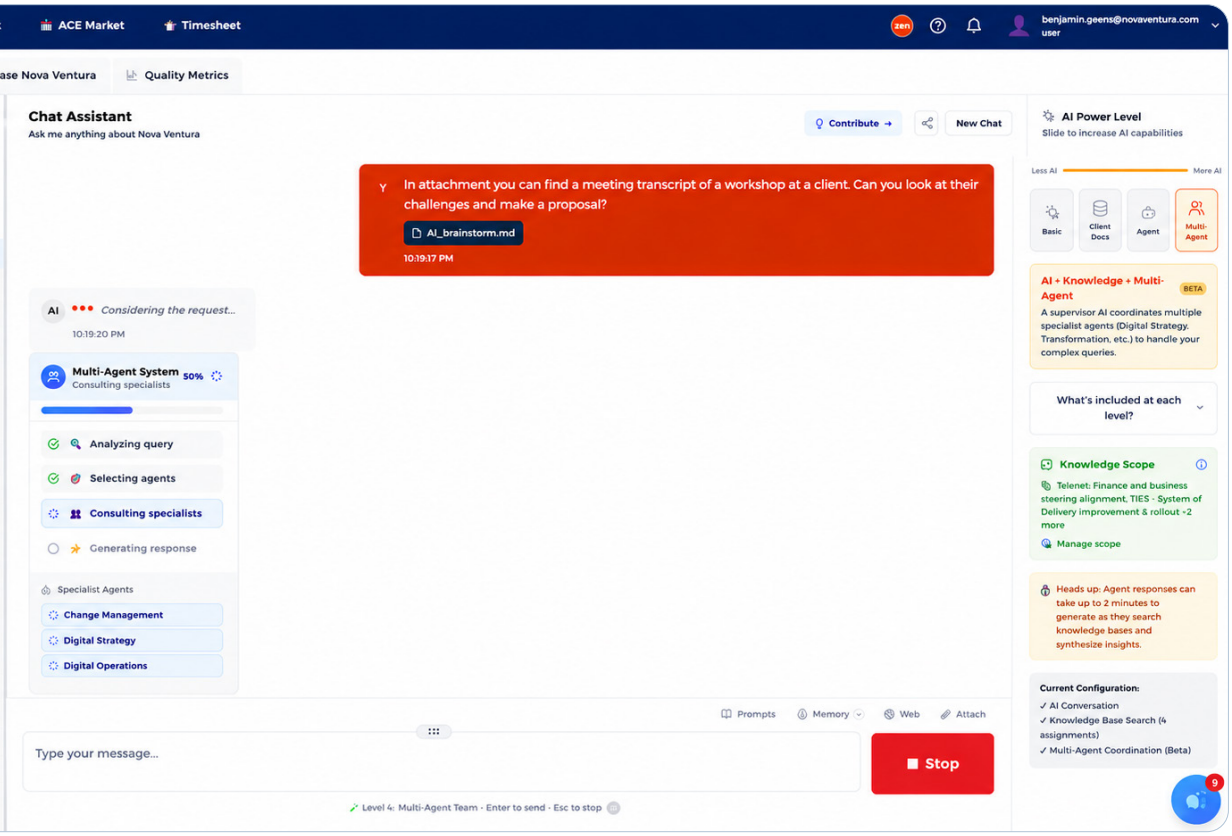
We set ourselves two objectives.

First, to rethink knowledge management now that AI can sit inside daily work, moving past document storage and search toward collective intelligence: a system that turns scattered individual knowledge into shared, usable guidance.

Second, to develop real competence in agentic coding: the field is moving fast, and as a strategic consultancy focused on digital transformation, we wanted hands-on experience we could carry into client work.

What came out of it was not a toy demo. It was a working internal AI platform, built part-time by a team of five: a single interface for the whole organization, combining a multi-model gateway, configurable custom agents, multi-agent orchestration, curated retrieval, feedback loops, and full operational visibility.

FIGURE – ACE Ventura chat



Building it forced us to make concrete choices about architecture, model routing, access control, and quality, while also surfacing real failure modes. Those decisions and learnings are documented in Parts 1 and 2.

But this paper does not stop at the experiment. Barbara Arnst, VP Transformation at one of our clients, Telenet (a 3,300-person telecom company operating roughly 150 teams), is navigating the same challenges at enterprise scale, in real conditions. Part 3 reflects what we are learning together: which patterns from a controlled experiment still hold when they are stress-tested inside a large, federated organization.

The paper is structured in four parts, followed by a set of leadership beliefs drawn from practice:

- Part 1: Building ACE Ventura: product decisions and learnings
- Part 2: Agentic coding: what we learned
- Part 3: A look into the future: how agentic systems will reshape organizations
- Part 4: What we are learning at enterprise scale, together with Telenet

Some of the more important discoveries were organizational. As the experiment progressed, it became clear that the challenges limiting AI value were not technical. They were structural: unclear decision ownership, standards that lived in people's heads rather than systems, and the absence of shared clarity at leadership level on what AI transformation actually requires. These patterns do not get easier at scale.

One conviction runs through everything that follows: the technology is the easy part. Value appears only when an AI-enabled way of working lands inside the organization, when it earns trust, gets used daily, and holds up over time. That is the harder, more interesting problem, and it is what this paper is really about.

Value appears only when an AI-enabled way of working lands inside the organization, when it earns trust, gets used daily, and holds up over time.

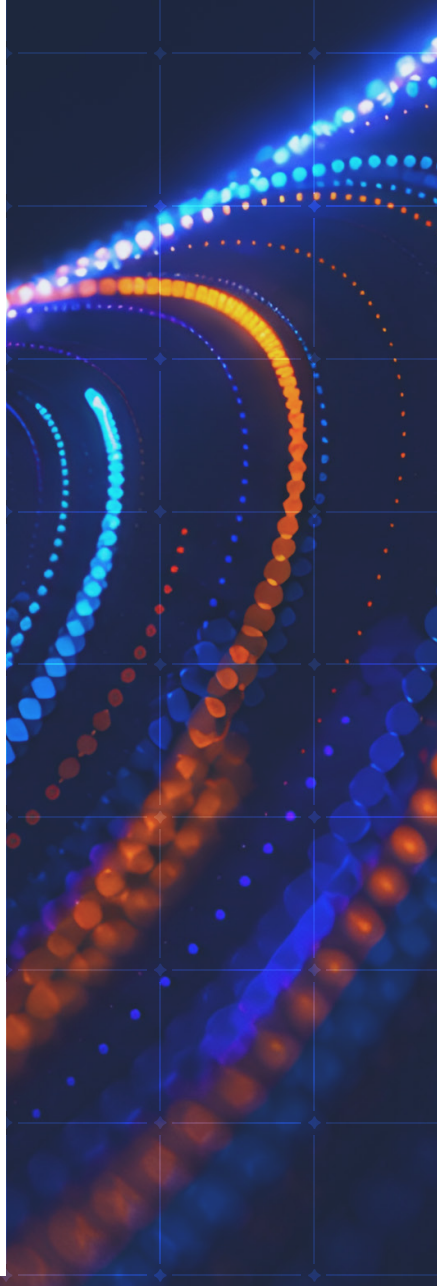
2

who this whitepaper is for



This document is for leaders and practitioners exploring how AI changes their organization, product, and delivery model. It is relevant for:

- CEOs and executives who want a real-world feel for agentic delivery and AI product adoption
- Digital transformation leaders and strategists
- Product owners and innovation teams building AI-enabled solutions
- CTOs, architects, and engineering managers evaluating agentic coding
- Leaders dealing with knowledge fragmentation and document sprawl
- Practitioners who see AI adoption as a change in how work gets done, not a tool rollout
- Enterprise transformation leaders navigating AI adoption at scale, where organizational prerequisites matter as much as technical ones



The goal is not to offer a universal framework. It is to present a grounded case and the lessons drawn from it, so that other teams can move faster and with fewer blind spots.

Introduction

why we ran this experiment

Nova is a 100-person consultancy group built around two complementary transformation capabilities. Nova Reperta helps organizations make strategy happen by translating ambition into clear direction, better ways of working, value creation, operational excellence, and sustainable change. Nova Ventura extends this into digital domain, helping organizations turn digital ambition into executable transformation through technology, data, architecture, operating model design, and change.

Together, they work in a context where knowledge and experience are core assets: delivery patterns, playbooks, approaches, reusable templates, client learnings, and proven transformation methods that can be rapidly tailored to each client's domain context.

Yet in many organizations, knowledge management still acts like a passive library. Documents get produced, stored, and then fade from use. Even when knowledge is "available," it often isn't available when you need it. And it rarely spreads beyond the people who wrote it. ACE Ventura is not a random starting point. We chose knowledge management deliberately because it is a strategic capability for our company. Our ability to consistently turn experience into reusable guidance, proposals, delivery approaches, templates, and delivery patterns is a core part of how we create value. Strengthening that capability directly affects productivity,

quality, and consistency across client work. For that reason, spending time and money on this experiment makes strong sense for both organizations.

In this era, where AI tools are easy to deploy, but much harder to make truly useful, the value depends on the knowledge architecture underneath: what is captured, cleaned, connected, governed, and made available in context. Without that foundation, AI does not transform knowledge work; it accelerates fragmented search, outdated answers, weak trust, and institutional memory loss.

For Nova Ventura, ACE Ventura became a deliberate experiment in turning passive knowledge management into active, AI-ready knowledge flows. This paper also explores the architecture behind that shift: from scattered knowledge assets to governed, reusable guidance that AI can safely access, reason over, and activate.

That is why we ran this experiment with two goals.

Our experiment was built to pursue both goals at once: change how our knowledge is used internally, while building real agentic delivery capability in a real product context.

First, we wanted to rethink the concept of knowledge management. Instead of treating knowledge as something people store and search for, we wanted to explore how knowledge can become usable guidance at the moment it is needed.

Second, we wanted to build an internal AI capability and learn what agentic coding really entails in practice. As agentic coding evolves quickly and starts to reshape how software can be built, we want hands-on experience so we can support our clients better in this transformation. This goes beyond "AI-assisted coding." With agentic workflows, agents can read goals, move through a codebase, propose changes, implement features, refactor, and generate tests.

4

Part 1

building ace ventura *product decisions and learnings*



PART 1

4.1 THE ORIGINAL USE CASE AND HOW IT EVOLVED

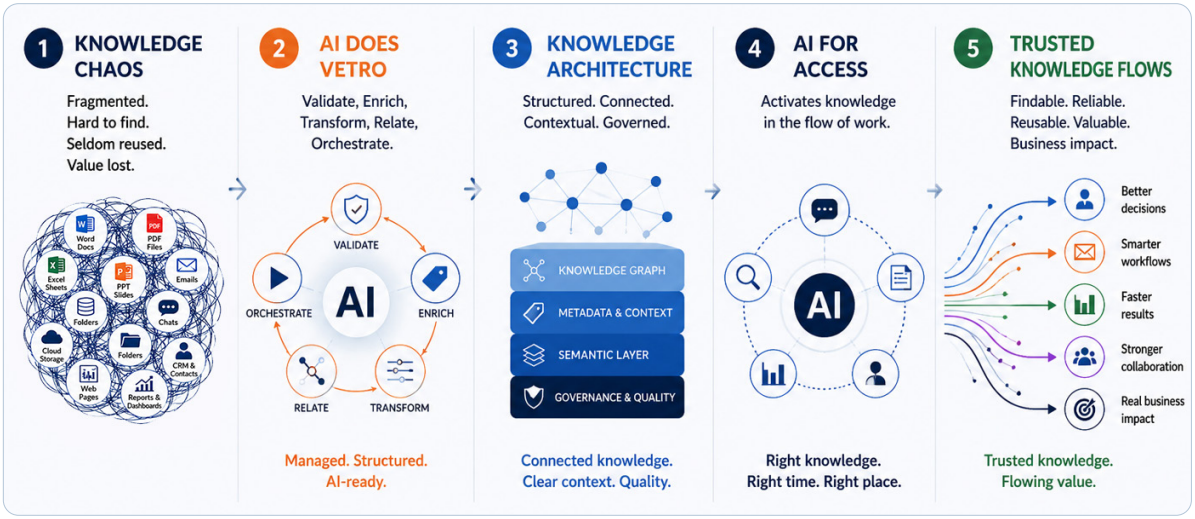
Before selecting a starting point, we looked at our knowledge management value stream end to end. The real problem was not a shortage of knowledge but an abundance of it: years of proposals, playbooks, delivery patterns, and client learnings scattered across people, documents, and tools. We wanted to identify where that fragmentation creates the most waste, where consultants spend disproportionate effort re-finding and re-assembling what already exists, and where consolidating it into a single, AI-curated base would have the biggest impact on how Nova Ventura and Nova Reperta operate.

We selected one scenario on purpose: generating a high-quality proposal quickly from client input, grounded in our internal knowledge and the way we actually deliver. This is a key engine of our business. It is a place where speed, quality, and consistency matter, and where a lot of effort typically goes into searching, assembling, and rewriting knowledge that already exists across people and documents. The goal was to shorten the path from a customer conversation to a tailored proposal draft by turning internal knowledge into actionable output.

We deliberately started small. The initial scope was clear and measurable: could the platform help a consultant take the context of a prospect or client conversation and turn it into a structured proposal that reflects our services, delivery approaches, and proven patterns?

Then agentic coding changed everything: shipping features suddenly felt almost free. Each improvement felt like a quick win, which made it easy to justify "one more addition." As a result, the platform grew fast. Over time, what started as a focused proposal accelerator expanded into a broader internal knowledge management platform: multi-agent orchestration, curated retrieval, a web UI above Teams and SharePoint, user and access management, logging, feedback loops, and admin-driven configuration.

The architecture that emerged from this expansion is shown below:



We are genuinely happy with the result, and above all with the delivery speed that became possible: features that once took weeks could be built in days. But we have to be honest about the flip side, because this turned out to be one of our most important learnings. Once shipping became so much faster and cheaper, we started building things nobody had asked for. We gold-plated. We added features because we could, not because anyone needed them, and several of them were never used. The constraint had quietly shifted from whether we could build something to whether we should, and weak validation meant we only noticed after the fact. That experience changed how we run ACE Ventura today. We put much stronger product management in place, and we installed a user feedback system with our colleagues so that what we build is validated against what they actually need, not against what is easy to ship.

✦ KEY LEARNING

Agentic coding multiplies speed, and it multiplies drift. Weak validation loops mean you can build the wrong thing faster.

4.2 WHY WE BUILT A CUSTOM PLATFORM INSTEAD OF RELYING ON OUT-OF-THE-BOX AI

We chose not to rely on an off-the-shelf experience like “just use Office 365 with embedded AI” as the main knowledge interface. Instead, we built a custom platform on top of what we already use: Teams and SharePoint.

This was not about reinventing everything. It was about control and “keeping our options open” in a fast-changing world of AI evolution.

Most organizationsface the same reality: their knowledge base is messy. Teams and SharePoint are often filled with outdated files, unfinished drafts, duplicates, and conflicting versions. When all of this is fed into retrieval, the AI produces answers that sound plausible but fail in subtle ways. Over time, trust erodes.

So we built a layer on top of Teams and SharePoint. They remain the collaboration interface and system of record, while our platform determines which content is ready to be used by the AI.

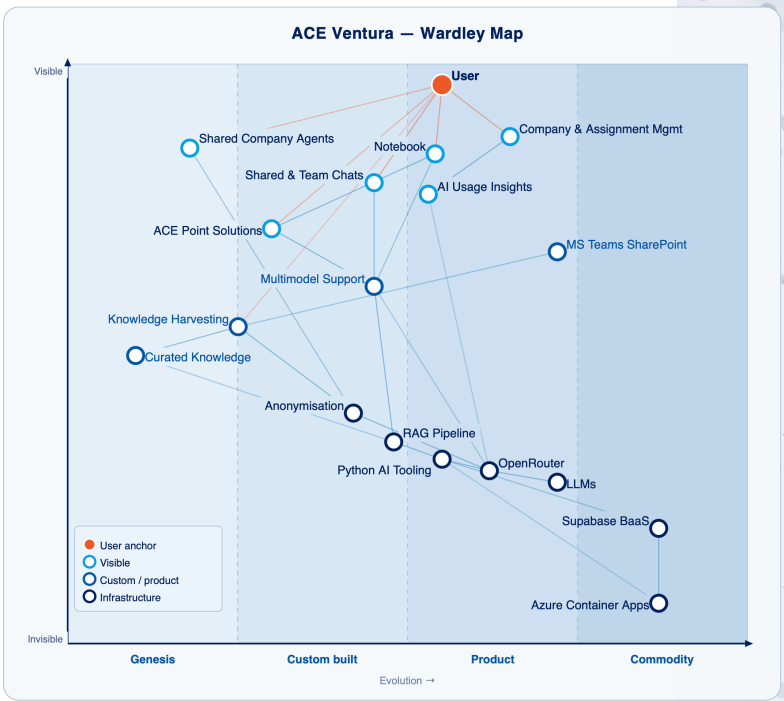
✦ KEY LEARNING

AI done wrong amplifies document chaos and accelerates messy output. The real work is building the knowledge layer that makes AI output trustworthy, reusable, and useful in context.

Design goals behind this choice were:

- ✦ Keep outdated or in-progress content out of answers
- ✦ Give agents stable, predictable context
- ✦ Control what becomes part of the knowledge base
- ✦ Keep trust high by reducing contradictory answers
- ✦ Curate the “latest and greatest” assets

To streamline the discussion on building yourself versus using available products we used a Wardley map. Our latest version, generated by Ace Ventura, is shown below.



4.3 KEEPING THE AGENT DEFINITION LAYER IN-HOUSE

We also chose to keep the agent definition layer in-house: prompts, skills, agent instructions, orchestration logic, and the way we encode delivery standards and our consulting approach into the system.

We treat that layer as a strategic asset. It captures how Nova Ventura thinks and works, how we turn knowledge into action, and how we keep consistency across engagements. If we outsource that layer to an external platform, we lose control over its direction and risk turning our differentiators into commodity settings.

Keeping it internal also lets us iterate quickly. As models change and our practices evolve, we can refine prompts, skills, and agent behavior without waiting on a vendor roadmap.

This also turned prompt and script management into a discipline of its own. The definition layer is not a handful of prompts; it is a growing library of versioned files that encode who we are and how we work: our branding profile, company identity, employee value proposition, public-speaking persona, messaging formulas and delivery patterns, among others. Treating these the way engineers treat code, with clear ownership, versioning, and review, is what keeps the system coherent as it grows. Without that discipline, the same files drift into conflicting copies and the agents start speaking with inconsistent voices.

✦ KEY LEARNING

Owning the agent definition layer preserves portability when models, platforms, and vendors change

4.4 TRUST AND QUALITY: WHY WE CURATE WHAT THE AI CAN USE

We implemented RAG (retrieval-augmented generation), but not as "connect it to everything." We built curated RAG: only selected documents can be used as retrieval context.

This is less a retrieval choice and more a governance choice. It puts a quality gate between the raw document pile and what agents can use. It also forces an operating model: someone must decide what enters the knowledge base, what stays, and what gets removed.

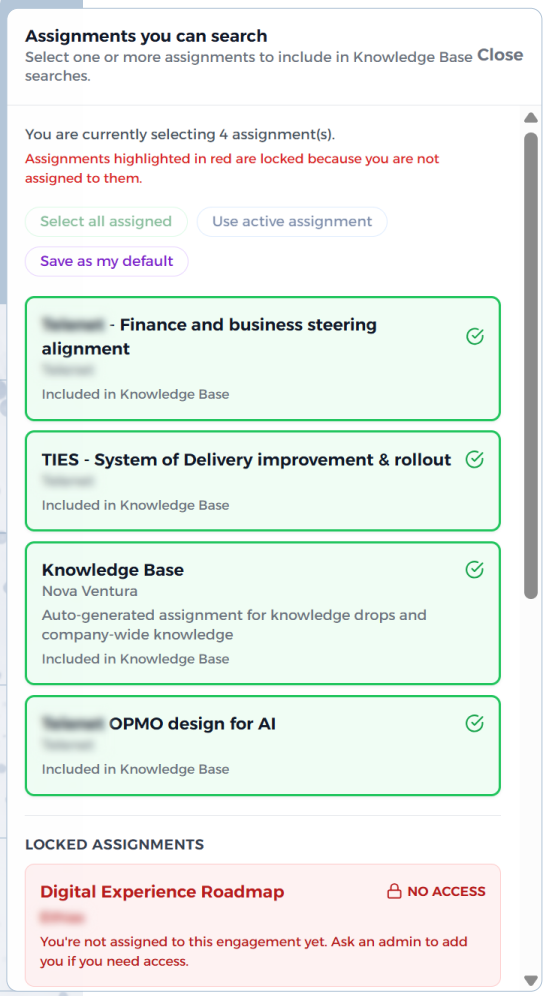
Curated RAG pushes a shift from "we store documents" to "we maintain a knowledge base." That's where the value shows up.

A useful contrast is:

- Full ingestion: fast start, uneven quality, trust drops over time
- Curated ingestion and disposal: slower start, steadier quality, trust rises over time as content is added and removed to keep the RAG clean and current

✦ KEY LEARNING

What you keep out of the knowledge base matters as much as what you put in. Every weak source the AI can reach lowers the quality, and the trust, of everything it produces.



4.5 CONFIDENTIALITY BY DESIGN: CONSULTANTS ONLY SEE WHAT THEY SHOULD

Security, access control, privacy, and client confidentiality matter as much in an AI setup as in any other system we build, maybe even more. An agent can find, connect, and expose information much faster than someone browsing through folders. Our core design rule for this was assignment-based access. Consultants should not be able to retrieve or browse documents across unrelated client engagements. The platform enforces strict boundaries: a consultant can only access documents tied to their active assignments, plus a curated layer of generic internal knowledge that is safe to reuse.

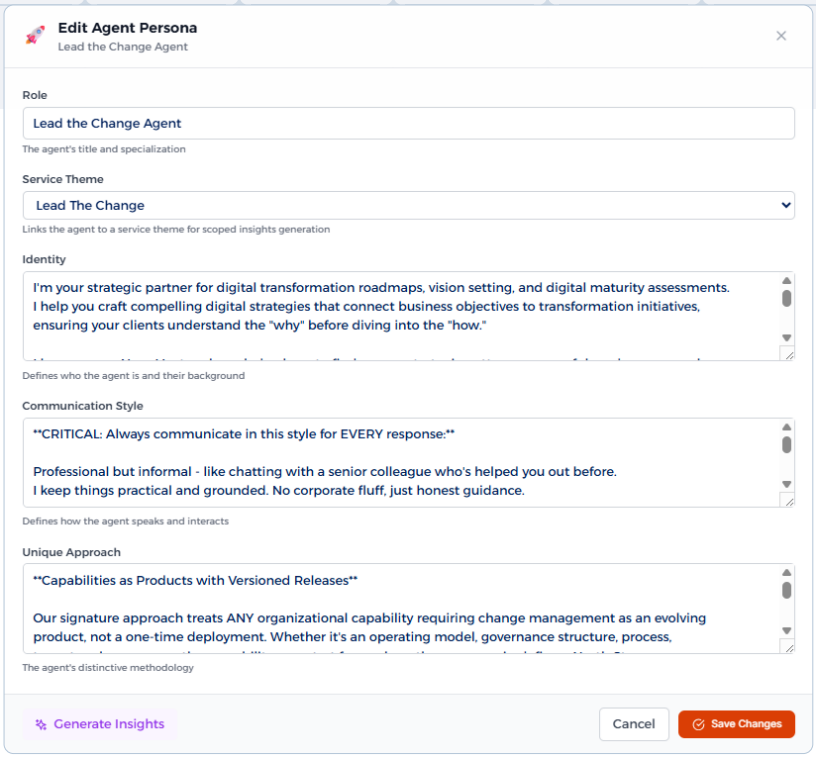
That generic layer includes what we want to reuse across contexts: best practices, delivery approaches, patterns, templates, and “war stories” written for reuse. Everything else stays assignment-scoped and only visible to the people working that engagement.

This is a confidentiality and security measure. It also improves answer quality. Many AI mistakes in knowledge systems come from mixing contexts that should never be mixed. If an agent can pull from another assignment, answers can become subtly wrong while sounding confident. We blocked that by design.

Technically, retrieval is filtered before generation. The agent does not see “everything.” It sees a permissioned subset of curated sources that fit the user and the context.

✦ KEY LEARNING

Access control matters more in an AI architecture, not less. An agent can surface and combine sensitive information much faster than any person browsing folders.



4.6 MULTI-AGENT ARCHITECTURE: ONE AGENT PER SERVICE OFFERING

We built the platform as a multi-agent system using LangGraph. The harder question was not the framework but how to divide the work between agents: what should each agent own, and where should the boundaries run? We tried several designs before it clicked. The one that proved stable was to organize agents around our service catalog, one agent per service offering, matching how Nova Ventura is already structured. In other words, our existing business structure became the architecture of the multi-agent system.

This is because most real consulting questions do not sit neatly inside one service. A client asking how to fund and govern an AI transformation is really asking a strategy question, a finance question, and an operating-model question at once.

This is exactly why the boundaries above matter: rather than build one agent that pretends to know everything, we let the bounded service agents work together, the same way we would pull specialists from different practices into one room. When a user asks a cross-service question, the platform calls the relevant service agents. Each agent answers inside its scope. A consolidation agent then merges those answers into one response.

This gave us a useful balance: cross-domain coverage without collapsing everything into a single blob.

✦ KEY LEARNING

The more agent boundaries reflect real-world structures, responsibilities and workflows, the more reliable they become.

4.7 MODEL ROUTING, COST CONTROL, AND OPERATIONAL VISIBILITY

Models change constantly. New ones appear, quality shifts, and prices move. Some models do better on reasoning, others on summarization, others on coding. Once AI sits inside daily workflows, model choice becomes an operating concern, not a background detail.

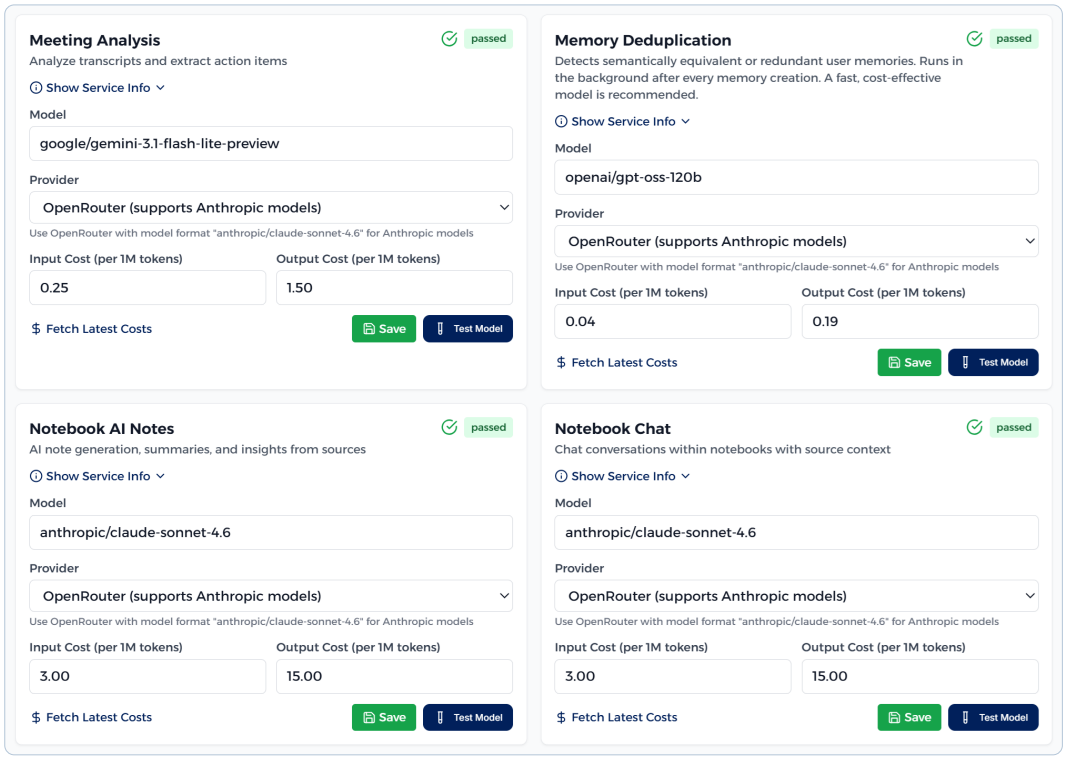
We knew from the start that we would need explicit control over which model is used for which capability, but it became a priority faster than we expected. The major model providers keep changing their pricing, so at production volume every token counts, and predictable cost management (FinOps) quickly became as important as the control itself. So, we built an admin screen to set model selection per capability through the UI. Under the hood, OpenRouter lets us switch models without reworking the platform. That makes it easy to respond when better models arrive, prices change, or a capability needs a different trade- off between quality and cost.

Model settings alone aren't enough. AI usage can scale quickly and unevenly, so you need visibility. We built usage tracking into the platform so we can see what is actually happening and adjust based on real numbers.

The platform tracks:

- Token use per feature, agent, and model
- Cost trends and FinOps signals across teams and engagements
- Latency and behavior across different model setups
- Carbon footprint indicators tied to AI usage patterns, which also feed into ESG reporting

This layer helps us tune not only answer quality and user experience, but also cost and responsible resource use. We wouldn't run a production platform without logs and monitoring. We treat AI-enabled workflows the same way.



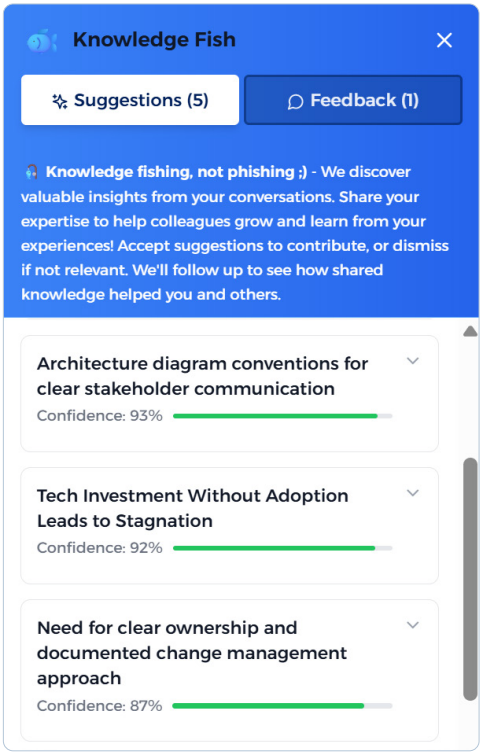
✦ KEY LEARNING

When AI becomes part of core operations, model choice, usage throttling and visibility must come together. That's how you keep control and keep the system healthy over time.

4.8 MAKING THE PLATFORM SELF-IMPROVING: LEARNING FROM USAGE

We do not want ACE Ventura to be a static knowledge library. Over time, knowledge decays, documents become outdated, and valuable insights stay trapped in conversations. A knowledge platform only stays valuable if it continuously reinforces what works and helps capture new learnings while people are actually doing the work.

That is why we implemented a small "fish" element hovering in the bottom-right corner of the screen. This feature continuously analyses the user's chat history with ACE Ventura and, via AI, extracts potential best practices, approaches, and war stories. The purpose is to lower the hurdle for our colleagues. Instead of expecting people to manually write down learnings after the fact, the platform proposes them proactively based on what was discussed.



We can also connect meeting transcripts to this functionality. When a community comes together to discuss challenges, cases, or missions, the fish uses those transcripts as additional context and proposes relevant best practices, approaches, and war stories. Users can then choose what is worth capturing and add it to the curated knowledge base, making it available for everyone.

KEY LEARNING

For knowledge management to last, knowledge capture must sit inside the workflow.

4.9 SOURCE LINKS AND FEEDBACK LOOPS

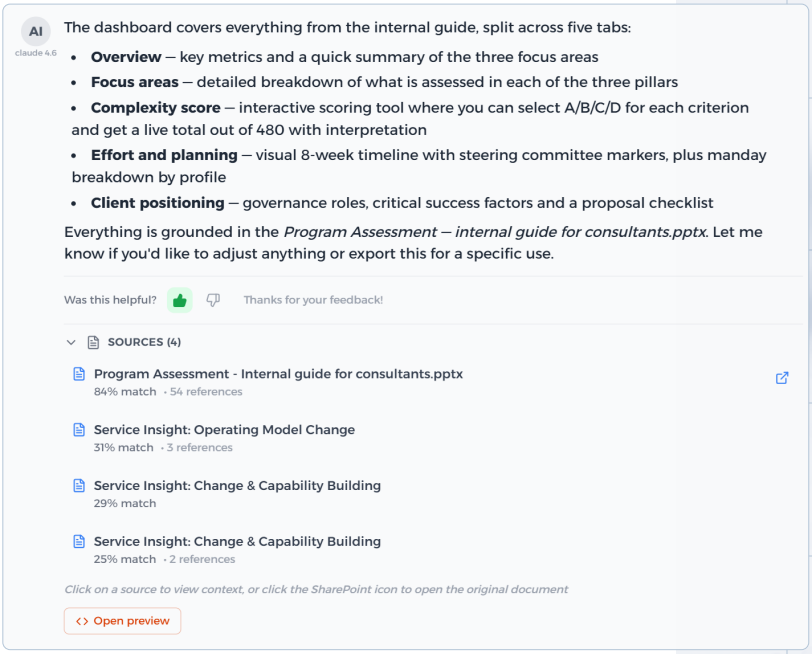
Trust is fragile in AI systems. People use what they can verify. If an answer sounds convincing but cannot be traced back to something real, users quickly stop relying on it, especially in a professional context where decisions and client work depend on correctness.

That is why ACE Ventura is designed to be transparent by default. Every answer includes source links so users can immediately see where the information comes from and open the underlying documents in Teams or SharePoint. This turns AI output from a “black box response” into something that can be checked, challenged, and trusted.

Feedback loops are the second part of the trust mechanism. Users can rate answers with a thumbs up or thumbs down. This creates an explicit learning signal that helps us improve the system over time. Strong answers get reinforced, while weak answers highlight gaps: missing knowledge, outdated sources, or content that should be curated differently.

Together, source links and feedback loops serve two purposes:

- They make answers safer to use, because users can validate them quickly
- They create a continuous improvement loop, because the platform learns what helps and what harms trust



KEY LEARNING

Feedback loops build trust, and trust drives adoption.

4

✦ KEY LEARNINGS AT A GLANCE

- ✦ Agentic coding multiplies speed, and it multiplies drift. Weak validation loops mean you can build the wrong thing faster.
- ✦ AI done wrong amplifies document chaos and accelerates messy output. The real work is building the knowledge layer that makes AI output trustworthy, reusable, and useful in context.
- ✦ Owning the agent definition layer preserves portability when models, platforms, and vendors change.
- ✦ What you keep out of the knowledge base matters as much as what you put in. Every weak source the AI can reach lowers the quality, and the trust, of everything it produces.

- Access control matters more in an AI architecture, not less. An agent can surface and combine sensitive information far faster than any person browsing folders.
- Agent boundaries that mirror real-world structure stay reliable.
- When AI becomes part of core operations, model choice, usage throttling and visibility must come together. That's how you keep control and keep the system healthy over time.
- For knowledge management to last, knowledge capture must sit inside the workflow.
- Feedback loops build trust, and trust drives adoption.

Part 2

agentic coding: what we learned

PART 2

5.1 TEAM SETUP: EXPLORATION HELPS, CONVERGENCE MATTERS

At the start of the experiment, we intentionally explored. The team was hybrid: five people, some with a technical background and some without. We set it up this way deliberately, to learn first-hand what agentic coding really means for people who do not usually write code, and whether it enables them to build. With that mix, we tested different models, workflows, and ways of collaborating with agents. That exploration gave us fast learning and helped us understand the capability landscape. It also created inconsistency. Different tools and models produced different coding styles, different assumptions, and different "interpretations" of what good looks like.

The key realization was that agentic coding does not remove the need for team alignment. It increases it. When code can be generated quickly, the cost of stylistic variance and architectural drift rises. The system becomes harder to reason about, pull request (PR) reviews, where one engineer checks another's proposed code changes before they are merged, get noisier, and integration becomes more painful than it needs to be.

What we changed during the experiment:

- We timeboxed experimentation and created a moment of convergence
- We agreed on one primary toolchain and one delivery rhythm
- We reduced parallelism on shared components
- We codified conventions earlier than we normally would, and we treated architectural specifications as a first-class artifact

✦ KEY LEARNING

Variability is expensive, even in a small team

5.2 MERGE PRESSURE: INTEGRATION PAIN SHOWS UP SOONER

One of the most surprising effects of agentic coding was how quickly integration pressure appeared. Throughput was high, which meant overlaps happened sooner: multiple people and agents working on the same files, refactors landing while new features were still being built, and large diffs becoming the norm. In a classic delivery model, teams often notice merge pain once the codebase becomes large or once delivery speed increases. With agentic coding, you get there immediately. The output volume increases, but the coordination model often stays the same. That gap creates friction.

How we worked in practice mattered as much as the tooling. We met twice a week for ninety minutes, and we ran a Teams channel where everyone posted every single thing they did, so the whole team could follow what was happening in near real time. We split the work between people on the scaffolding and harness and people on the functional features, and refactors were shared immediately so nobody built on a moving foundation.

The diagram below shows the architecture and tech stack behind ACE Ventura: the layers from the React web app down to the Supabase data platform, and the engineering platform we built on.



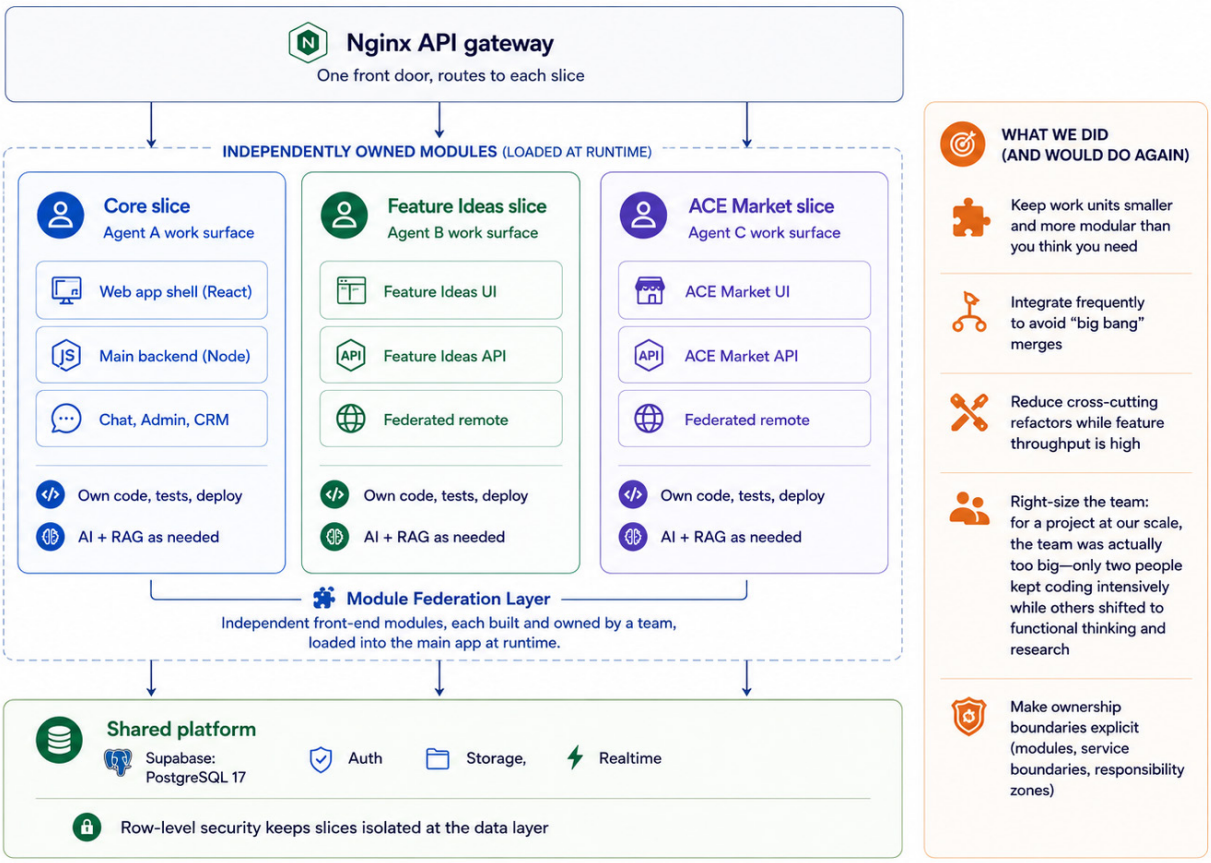


FIGURE – ACE Ventura architecture and tech stack

One move helped more than any other: over time we deliberately made the application more modular, so that more people could contribute in parallel without getting in each other's way. The Module Federation layer in the diagram is a concrete example: independent front-end modules, each owned and built on its own, then loaded into the main app at runtime. This lowered the barrier to participation, including for the less technical members of the team, because someone could build and ship a self-contained module without needing to understand the whole codebase. Modularity turned out to be not just an architecture choice but the thing that let agentic coding scale across the team.

Why this happens:

- Agents produce changes fast, but they don't naturally coordinate across parallel work
- Large diffs are common because agents tend to "solve the whole problem"
- Refactoring becomes cheap, so people do it more often, increasing collision risk

What we did (and would do again):

- Keep work units smaller and more modular than you think you need
- Integrate frequently to avoid "big bang" merges
- Reduce cross-cutting refactors while feature throughput is high
- Right-size the team: for a project at our scale, the team was actually too big, and we ended up with only two people still coding intensively while the others shifted to functional thinking and research
- Make ownership boundaries explicit (modules, service boundaries, responsibility zones)

✦ KEY LEARNING

More throughput increases coordination needs. It doesn't erase them. And as a few people do most of the building, the gap in who actually understands the application widens, which becomes its own coordination problem.

5.3 THE AGENT MEMORY PROBLEM: CONTEXT DROPS BETWEEN SESSIONS

The most frustrating limitation we hit was memory. Agents are powerful inside a session, but they do not reliably carry context across time. Architecture rules, quality expectations, naming conventions, and even design intent fade quickly once the context window is exceeded or a session ends. That led to repetition: reteaching patterns, restating rules, and correcting drift.

This is not a small usability issue. It becomes a scaling constraint. If you keep rebuilding context manually, productivity drops and the codebase becomes less coherent over time.

How we worked around it:

- We used persistent instruction files and tool-specific skills
- We wrote down conventions explicitly instead of relying on "team memory"
- We embedded architecture decisions in documentation and reference files
- We repeatedly forced alignment through review and correction loops

What we discovered:

- Agents don't "remember standards" unless standards are encoded
- Every missing rule becomes a future inconsistency
- The more complex the system becomes, the more expensive memory loss becomes

KEY LEARNING

Persistent memory is core infrastructure for scaling agentic development.

5.4 THE ECOSYSTEM MOVES TOO FAST: BALANCING NOVELTY AND DELIVERYS

The agentic coding landscape evolves at extreme speed. New models appear, new IDE workflows emerge, and new orchestration patterns become available almost weekly. For a curious team, it is very tempting to chase the newest capability. But continuous novelty comes with a price: relearning, rebuilding, and rewriting.

We experienced this directly. If everyone continuously integrates new tools or patterns, progress slows down even while activity feels high. You end up in permanent transition.

What we did that worked:
We separated into two modes of work.

- **A stable delivery mode** with a fixed stack and consistent workflow
- **A lightweight exploration mode** where new capabilities were tested in small trials

That created a natural R&D rhythm: the team keeps learning, without constantly destabilizing delivery.

✦ KEY LEARNING

Novelty needs a cadence, or it becomes a tax.

5.5 FEATURE CREEP AMPLIFIED: BUILDING BECOMES ADDICTIVE

Agentic coding makes building feel cheap. That changes product behavior. Features that would normally be deprioritized suddenly feel like quick wins. The platform grew far beyond the original idea, not because anyone made a strategic decision to expand scope, but because it was easy to keep going.

The result is a double-edged sword. We are proud of what we built, but we also saw how easy it becomes to outpace validation. If user feedback loops are weak, agentic coding can accelerate the building of a beautiful system that is not yet anchored in real usage and outcomes.

What we would do differently next time:

- Define an explicit MVP boundary and defend it
- Introduce “validation gates” after each major slice
- Force early user interaction, even if the product feels unfinished
- Treat scope control as part of engineering discipline

✦ KEY LEARNING

Product discipline matters more when building feels cheap.

5.6 QUALITY: STANDARDS MUST BE WRITTEN INTO THE SYSTEMS

Agentic coding can generate impressive code, but it does not guarantee quality. Without guardrails, speed becomes risk. Architecture drifts, testing becomes inconsistent, and regressions appear faster than humans can catch them.

This matters even more in agentic delivery because the volume of change increases. It is also the natural next step after the memory problem in section 4.3: writing standards down is what stops them being forgotten, but at agentic speed that is not enough on its own. When agents can refactor large parts of a system quickly, manual review alone cannot be the main protection mechanism. The encoded standards have to be enforced automatically, by the system, on every change.

Concretely, this is how we wired it, and the choices are pragmatic rather than permanent. The field is shifting so fast that betting everything on one setup would be naive, so we made deliberate, reversible choices. We picked Claude Code as our main agentic coding agent and lean into its system: CLAUDE.md, skills, hooks, and configuration. That is where we encode how we want the agents to work. But we are clear-eyed that this layer is not deterministic: the same instructions do not always produce the same result. So, wherever we can, we mirror the same checks in our CI pipeline. Claude Code runs on each developer's local machine, which means its guardrails are only as consistent as each setup, whereas the CI runs the same checks for everyone, at team level. The diagram below shows these two halves: on the left, the Claude Code setup that guides the agents; on the right, the CI/CD pipeline that enforces the same standards on every change. We also regularly cross-check what Claude proposes against other models such as Codex, to get a second opinion rather than trusting a single model's judgement.

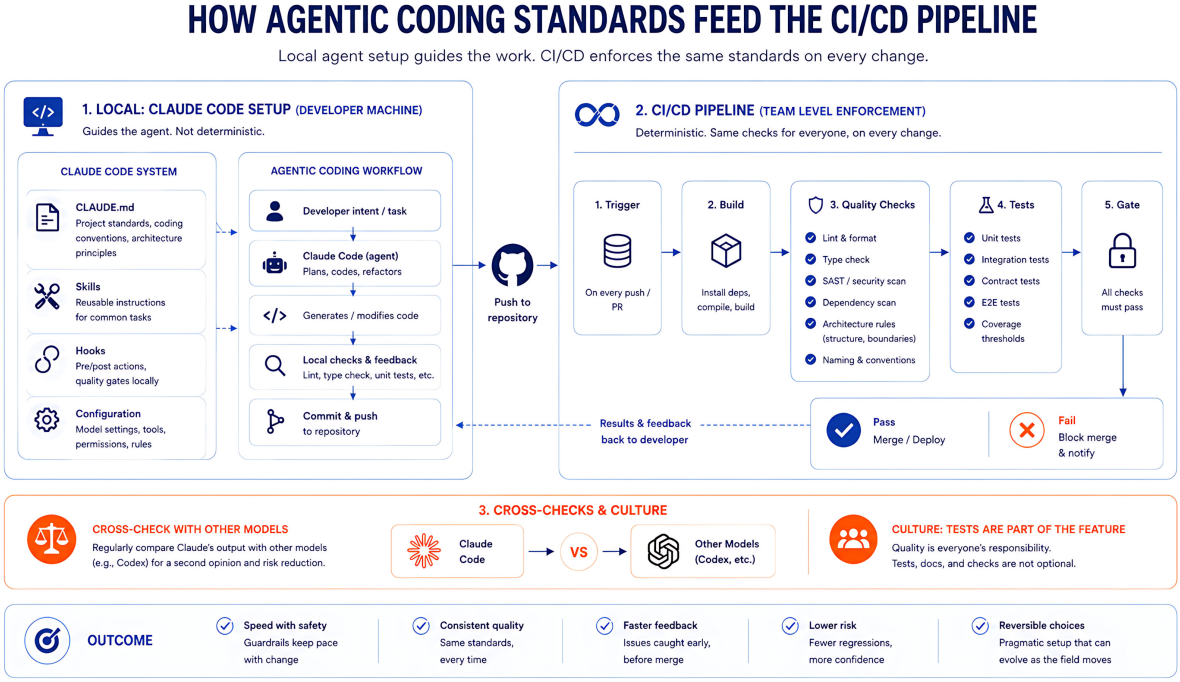


FIGURE – how agentic coding standards feed the ci/cd pipeline

What worked for us:

- A CI/CD pipeline running on every change
- Automated tests as the primary feedback loop
- Clear standards for structure, naming, and architecture conventions
- Claude Code as our main agentic coding agent, with its system (CLAUDE.md, skills, hooks, and configuration) used to encode how we want the agents to work
- The same quality checks mirrored in CI, so guardrails hold at team level even though Claude Code runs locally and is not deterministic
- Regular cross-checks of what Claude proposes against other models such as Codex, for a second opinion
- A culture of "tests are part of the feature" rather than an afterthought

✦ KEY LEARNING

When the agent is non-deterministic, the pipeline becomes your source of truth. The standards you can rely on are the ones enforced automatically on every change, not the ones living in a local setup.

✦ KEY LEARNINGS AT A GLANCE

- ✦ Variability is expensive, even in a small team.
- ✦ More throughput increases coordination needs. It doesn't erase them. And as a few people do most of the building, the gap in who actually understands the application widens, which becomes its own coordination problem.
- ✦ Persistent memory is core infrastructure for scaling agentic development.
- ✦ Novelty needs a cadence, or it becomes a tax.
- ✦ Product discipline matters more when building feels cheap.
- ✦ When the agent is non-deterministic, the pipeline becomes your source of truth. The standards you can rely on are the ones enforced automatically on every change, not the ones living in a local setup.



Part 3

a look into the future

how agentic systems will reshape organizations

PART 3

The pace of change in agentic systems is so high that a true "look into the future" is, by definition, imperfect. In this space, certainty ages quickly. New models, new orchestration approaches, and new delivery patterns emerge faster than organizations can stabilize their playbooks. So, the purpose of this section is not to predict winners or to present a definitive blueprint.

Instead, we share the insights that already feel structurally true based on what we have learned so far: through our ACE Ventura experiment, through our broader transformation experience, and through the way we navigate this shift together with clients such as Telenet. Our approach is to combine practical experimentation with a systems view on operating model change. We lean on proven transformation patterns, while staying honest that this domain is still evolving and must be explored in short learning loops.

In other words: we may not be able to forecast the future with precision, but we can already see where the direction is heading and how organizations can move through this complexity with more control and less friction.

6.1 FROM IDE COPILOTS TO AGENTIC SYSTEMS ACROSS THE LIFECYCLE

In our experiment, agentic coding mostly happened through Claude Code in the IDE. A single agent could read goals, navigate the codebase, implement changes, refactor, and generate tests. For many teams today, this is the most accessible entry point into agentic development: it fits naturally into existing workflows and toolchains.

But as we were building ACE Ventura, we also began to see that this approach may already be an intermediate stage. The field is rapidly moving beyond "agentic coding in the IDE" toward orchestration-driven systems where the IDE is no longer central, and in some setups not used at all. In this direction, software delivery becomes an agentic workflow executed by an orchestrator that coordinates many specialized agents. Humans define intent, constraints,

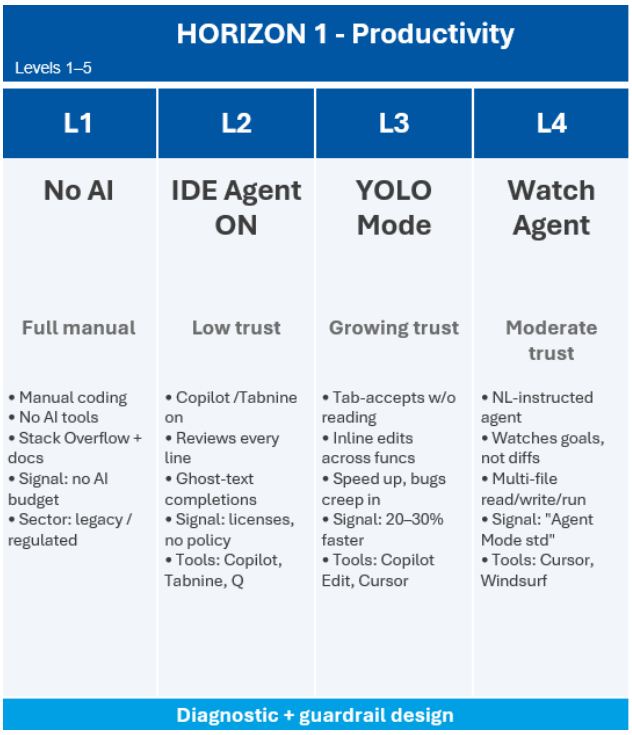


FIGURE
maturity model, from ide-based agentic coding toward ide-less multi-agent orchestration across the sdlc

acceptance criteria and architectural specifications, while the agentic system executes large parts of the SDLC as a coordinated process.

Work by people like Steve Yegge and Geoffrey Huntley illustrates this shift: agentic delivery moving toward multi-agent orchestration systems that operate across repositories, environments, and pipelines without relying on the developer's IDE as the primary interface.

The shift happens in stages rather than all at once. The maturity model below sketches that progression, from agentic coding inside the IDE, where a single agent assists a developer, toward IDE-less, orchestration-driven delivery, where a coordinator runs many specialized agents across the lifecycle. We see today's IDE-based setup, which is where our experiment sits, as an early stage on this path rather than the end state.

HORIZON 2 - Parallelisation Levels 6–8			HORIZON 3 - Systemic Transformation Levels 9–11				
L5	**L6**	**L7**	**L8**	**L9**	**L10**	**L11**	
CLI-First Agent **High trust ⚠** • Issue → async PR • IDE = review only • Agent runs in CI sandbox • Signal: tickets to AI • Tools: Claude Code, Devin	**Several Agents** **Ad-hoc coord.** • 2–5 parallel agents • Dev multiplexes • No shared queue • Signal: "chaotic but fast" • Risk: spec drift, conflicts	**10+ Agents Manual** **Chaos risk** • 10+ agents manual • Devs = bottleneck • Merge storms, dup code • Signal: collapse at 8+ • Risk: token blow-up	**Custom Orch.** **Shared queue** • Shared task queue • Coord + checkpoints • Verification pipeline • Signal: internal tooling • Tools: Gas City, Intent.ai	**First Factory** **AI-run WF** • 1 prod AI workflow • Code-review / IR / CI • Human monitors only • Signal: "runs unattended" • NV: pilot factory design	**Portfolio Factory** **Human bottle** • Packs per domain • Dozens running 24/7 • Engineer = control plane • Signal: "15+, no overview" • NV: portfolio steering	**Factory Builder** **AI manages AI** • AI manages AI teams • Area manager packs • Reliability per domain • Signal: AI runs eng org • NV: advisory, audit	The unknown
Architecture + Operating Model + Governance							

In these systems, the "unit of work" is not a developer prompting an IDE assistant. The unit of work becomes a delivery objective that gets decomposed into tasks, assigned to multiple agents, executed in parallel, validated, and then packaged into reviewable outputs.

A typical orchestration flow can involve agents responsible for:

- Clarifying scope, constraints, and intended outcomes
- Breaking down work into tasks and sub-tasks
- Modifying code across modules or services in parallel
- Generating and extending automated tests
- Running builds, quality gates, and CI pipelines
- Analyzing failures and iterating until checks pass
- Producing documentation and change notes
- Creating pull requests, release artifacts, or deployment-ready outputs

This evolution matters because it changes what organizations need to optimize for.

With IDE-based agentic coding, the perceived impact is often "developer productivity." Teams become faster, but the delivery model remains recognizable. Engineers still drive the workflow, and the agent amplifies execution.

With IDE-less multi-agent orchestration, the impact shifts from individual acceleration to system-level acceleration. The organization is no longer adopting a tool. It is adopting a new delivery mechanism. That raises the stakes: throughput can rise dramatically, but only if governance and control mature at the same pace.

In practice, this creates **several predictable consequences**.

- **First**, intent becomes the bottleneck. When execution is cheap and easy to parallelize, the hardest part becomes deciding what should be built, how it should behave, which standards must hold, and when it is “good enough” to ship.
- **Second**, integration and coherence become harder. Orchestrated agent swarms can produce large volumes of change quickly. Without strict boundaries, explicit architecture rules, and automated enforcement, the output becomes inconsistent and the system becomes harder to maintain.
- **Third**, quality enforcement becomes non-negotiable infrastructure. Automated tests, CI/CD, policy checks, and traceability are not “good practices.” They become the control layer that keeps an increasingly autonomous delivery system safe.
- **Finally**, this shift will change roles. Engineers spend less time typing and more time steering: defining constraints, reviewing outcomes, validating correctness, managing risk, and shaping system evolution. Organizations that treat this as “faster coding” will miss the real impact.

Our experiment therefore sits at an important transition point. It proved what is already possible today with agentic coding in a real codebase. At the same time, it made clear that the direction of travel is toward IDE-less orchestration systems that can execute large parts of the SDLC as coordinated, multi-agent workflows.

6.2 THE IMPACT ON OPERATING MODELS: ROLES, RESPONSIBILITIES, AND DECISION FLOWS

Agentic systems do not only change how software is built. They change who can build software, what "building" means, and how organizations decide what gets done. The operating model impact will likely be larger than most teams expect, because these capabilities push software creation beyond engineering departments and into business domains.

We are entering a world where non-technical roles can create working applications that go far beyond MVPs. At first, this will look like "business teams prototyping faster." Over time, it will look like business teams owning real production-grade capabilities, supported by governance and engineering enablement rather than traditional implementation dependency.

This shift forces organizations to revisit three fundamentals: decision flow, role design, and the boundary between product intent and technical execution.

6.2. 1 The builder profile expands beyond engineering

Historically, software creation was constrained by scarce technical implementation capacity. Product leaders and business teams could define intent, but engineering held the execution keys. Agentic systems reduce that constraint. When a large part of implementation can be automated or orchestrated, more people can move from "requesting software" to "creating software."

This does not remove engineering. It changes engineering's role from primary builder to system enabler and governor. Engineering becomes responsible for the environment that makes safe creation possible: architecture, platforms, quality automation, security controls, cost visibility, and delivery guardrails.

6.2.2 Decision flow changes: intent becomes the bottleneck

As execution becomes cheaper, the bottleneck shifts upstream. The limiting factor becomes the quality of intent: clarity of desired outcomes, constraints, trade-offs, and acceptance criteria.

In an agentic world, the most valuable skill is not writing code. It is defining what good looks like in a way that can be validated. That means product thinking, domain clarity, and decision-making discipline become central.

This also changes governance. Many organizations are optimized for scarce build capacity, with long intake processes and heavy prioritization layers. As delivery accelerates, organizations must avoid turning faster execution into chaos. They need decision flows that are faster, but also clearer and more explicit.

6.2.3 Team design: do equal team sizes still matter?

Many operating models assume relatively stable team shapes: cross-functional teams with comparable capacity and a predictable ratio of business roles to engineering roles. Agentic systems will challenge that assumption.

If implementation capacity expands through agents, a team may not need the same number of engineers to deliver the same output. That does not mean teams become smaller everywhere. It means "capacity" becomes more elastic and more dependent on governance maturity.

We may see several patterns emerge:

- Smaller product teams that can deliver more, because agentic execution amplifies implementation
- A shift from fixed team structures to more dynamic "mission teams" assembled around outcomes
- Engineering capacity moving into platform and enablement functions, rather than direct feature delivery
- New specialist roles focused on orchestration, quality enforcement, and AI operational control

Instead of optimizing for equal team sizes, organizations will need to optimize for coherent ownership, safe autonomy, and fast validation loops. The question becomes less "how many people are in the team" and more "what does this team own, what can it change safely, and how is quality enforced."

6.2.4 The limit of shipping speed: the customer must absorb change

Agentic systems can radically increase output. But output is not the same as value. Shipping faster does not automatically create advantage if users and customers cannot absorb the pace of change.

Every organization faces a constraint that is often ignored in engineering discussions: adoption bandwidth. Customers have learning curves, change fatigue, and operational dependencies. Too much change creates friction, even if the changes are "good."

This is where operating models must evolve. Teams will need to plan for:

- Change absorption and adoption, not only feature delivery
- Release pacing based on learning and customer readiness
- Product communication as a core capability, not an afterthought
- Operational readiness as part of "done"

In other words, faster building forces a stronger product discipline around what gets released, when, and how customers experience it.

6.2.5 Competitive dynamics: when features can be copied in weeks

A second constraint is market dynamics. If agentic delivery allows competitors to copy features in weeks, the advantage of individual feature novelty shrinks. Organizations that relied heavily on shipping "the next feature" as differentiation may see diminishing returns.

This doesn't mean features stop mattering. It means sustainable advantage shifts toward elements that are harder to replicate quickly, such as:

- Integrated end-to-end customer experience, not isolated features
- Trust, reliability, and consistent service quality
- Ecosystem and platform lock-in (in a positive sense: integration value)
- Brand credibility and relationship strength
- Distribution advantage and customer intimacy
- The ability to learn faster than competitors, not only build faster

This also changes how customer loyalty might evolve. If switching costs drop and competitors match features quickly, loyalty becomes less about feature leadership and more about experience, confidence, and outcomes delivered consistently over time.

6.2.6 What organizations will need to redesign

Agentic systems will not simply slot into today's operating models. Organizations will need to redesign how responsibilities and decision flows work when execution scales.

Some operating model adjustments that will likely become necessary:

- Stronger product intent mechanisms: clear goals, constraints, acceptance criteria
- Governance that enables speed safely: automation, policies, boundaries, traceability
- Platform enablement: teams building guardrails so more people can build responsibly
- Release strategy that balances delivery speed with adoption speed
- Clear accountability even when execution is automated: ownership stays human

6.3 CHANGE MANAGEMENT BELONGS IN THE OPERATING MODEL DISCUSSION

These shifts cannot be delivered through tooling alone. They require adoption, and adoption is harder than most operating model discussions acknowledge.
People need to trust the system, change habits, and work in new ways. A strong AI platform fails if it is not used, not trusted, or if workflows stay the same around it.

That is why change management is not a separate workstream to run in parallel with the technical transformation.
It belongs inside the same conversation as operating model redesign, from the start.
What that looks like in practice, the leadership beliefs and organizational choices that determine whether adoption sticks or stalls, is what the next section explores, through the lens of our ongoing collaboration with Telenet.

Part 4

what we are learning at enterprise scale, together with Telenet

By **Barbara Arnst**



PART 4

ACE Ventura gave us a controlled environment to learn fast. At roughly 100 people and a single delivery context, it was possible to hold the full system in view. But the organizational challenges we encountered (decision rights, codification, leadership alignment) do not get easier at scale. They get harder and more consequential.

That is why this paper does not stop at the experiment. Telenet is a Belgian telecom company operating ~150 teams across a federated delivery model with multiple brands. It is the kind of organizational context where patterns discovered in a controlled experiment get stress-tested in ways that matter, and where Barbara Arnst, as VP Transformation, is navigating exactly the challenges this paper describes, at enterprise scale and in real conditions.

We approached this as shared discovery between Telenet and Nova Ventura rather than applied consulting. What follows reflects what we are learning together.

7.1 THREE DISCOVERY TRACKS WE ARE WORKING ON TOGETHER

The work combines three connected tracks. Each addresses a different layer of the same challenge: how do you capture AI's potential when the organizational prerequisites are still being built?

TRACK 1

Proving and scaling AI-powered SDLC automation

The ambition is concrete: prove that AI-powered automation can be applied meaningfully across the full software delivery lifecycle, from requirements and design through coding, testing, and release, and then scale what works. The work explores how agentic workflows can reduce manual effort, shorten end-to-end lead times, improve quality, and lower the cost of value delivery. This is not a productivity experiment in isolation. It is the proving ground for understanding where agentic delivery genuinely changes software development, and what organizational prerequisites need to be in place to maximize what is achievable.

TRACK 2

Redesigning the operating model for agentic delivery

Agentic coding is not a faster IDE. It changes who can build, who decides, and how organizations keep control when execution accelerates. We are working through what this means in practice: how ownership boundaries and decision rights need to shift, where existing governance structures create bottlenecks at agentic speeds, and how team design needs to evolve when delivery capacity becomes more elastic. The goal is not a new framework. It is a set of concrete operating model adjustments validated in real delivery contexts.

TRACK 3

Building a knowledge layer that stays trustworthy at enterprise scale

The knowledge management insights from ACE Ventura translate directly, but the enterprise context introduces challenges a consultancy experiment never surfaced: a far larger document landscape, more complex access control requirements, and a knowledge base spanning multiple domains, brands, and delivery contexts. Together, we are exploring what curated knowledge governance looks like at this scale: how to prevent context contamination, how to design trust mechanisms that hold under real usage, and how to make knowledge capture a natural part of delivery rather than an administrative burden layered on top.

These three tracks are not independent workstreams. SDLC automation generates the throughput that makes operating model gaps visible. Operating model clarity is the prerequisite for knowledge practices to scale. And trustworthy knowledge is what keeps agentic delivery coherent as speed increases. They are designed to reinforce each other.

7.2 WHAT WE ARE LEARNING TOGETHER SO FAR

The patterns from ACE Ventura hold at enterprise scale. But the blast radius is different. Moving from a 100-person consultancy to a 3,300-person telecom company does not just amplify the challenges; it reveals new ones that only surface when AI runs into real organizational complexity. Two learnings stand out so far.

Clarity on scope, priorities and decision ownership is a prerequisite, not a byproduct

AI transformation forces organizations to make explicit what they have always been able to leave implicit. Strategic trade-offs (speed versus safety, autonomy versus control) have historically been managed as they arose and resolved case by case. Team boundaries and decision ownership follow the same pattern: everyone roughly knows who owns what, and slow execution creates enough slack to negotiate the grey zones when they surface. Agentic delivery removes that slack entirely. Two things become immediately visible that were previously manageable in the background: differing leadership priorities on what actually matters, and the absence of clear ownership at execution level.

Both show up quickly at enterprise scale. Leadership goals (speed and safety) were both legitimate but required explicit scenario-level alignment to translate into coherent guidance at execution level. That alignment work is easy to underestimate. At the same time, teams working in well-defined contexts with explicit ownership moved and learned faster than teams operating in fuzzier boundaries, regardless of technical capability.

The two problems are connected: when leadership trade-offs are unresolved, decision ownership below cannot be clean either. Ambiguity compounds downward.

The shift we are making:

- Aligning leadership on acceptable trade-offs at scenario level, not just in principle: the abstract agreement is rarely the missing piece
- Defining decision rights explicitly at three levels: strategic AI direction, capability adoption, and operational execution choices
- Mapping ownership boundaries before scaling AI practices, not after: retrofitting clarity onto accelerating delivery is significantly harder than designing it upfront
- Redesigning governance cadences to match delivery speed: monthly review boards are structurally incompatible with weekly delivery cycles

✦ KEY LEARNING

Agentic delivery does not create misalignment or unclear ownership. It removes the slack that allowed both to remain invisible. What was manageable ambiguity at low speed becomes an immediate blocker at high speed.

Tacit knowledge is invisible to agents

Every standard, constraint, or quality rule that is not explicitly encoded disappears from the system the moment a human steps back. This is not a tooling problem; it is a structural property of how agentic systems work. Agents do not carry context between sessions. They do not absorb patterns through osmosis or self-correct through collaboration. They work with what is explicitly available to them, and nothing else.

Many high-performing enterprises share a people-oriented, action-driven culture with a strong bias toward getting things done. Such a culture is a genuine strength and is what makes organizations like Telenet move faster than more bureaucratic ones. Documentation and codified standards feel like overhead, and in a human-centric delivery model, that intuition is not entirely wrong. Experienced people remember context, propagate patterns informally, and course-correct through conversation. That system functions because humans are good at reading between the lines. Agents are not. The pattern we have seen is a slow degradation: early outputs were coherent because experienced engineers were closely involved. As coding agents took on more, drift accumulated. Quality rules that everyone "knew" were not being applied because no agent had been told about them explicitly.

Architectural constraints that were obvious to the team risk becoming invisible to the system.

This is the same constraint we encountered in the ACE Ventura experiment; section 4.3 describes how agents fail to carry context between sessions at the individual delivery level. What the Telenet context reveals is that the same structural problem exists at organizational scale: standards that live in people's heads and cultures rather than in systems disappear just as surely as context that exceeds a session window

The shift we are making:

- Treating codification as a first-class engineering capability with dedicated time in delivery rhythms, not an administrative task fitted around "real" work
- Making quality rules executable and embedded in tooling rather than advisory
- Building the connection between absent standards and specific rework visible to teams, so the cost is experienced rather than theorized

✦ KEY LEARNING

Tacit knowledge is not lost in agentic delivery; it simply never enters the system in the first place. If it is not encoded, it does not exist.

7.3 PERSPECTIVE OF BARBARA ARNST FROM TELENET

The preceding sections cover what we built, what broke, and what we learned together. Before closing, I want to shift register, away from execution detail and toward what I believe determines whether a transformation of this magnitude succeeds or fails at the leadership level. These are not methodology recommendations. In my experience, these are the beliefs that separate transformations that build lasting capability from those that create expensive noise.

Belief 1:

You can't export clarity you don't have

The instinct when facing AI disruption is to move fast and show results. That instinct may cost you. Not because speed is wrong, but because you cannot scale practices built on a fuzzy foundation. What looks like slow progress to build that foundation up front is actually needed to prevent fragmentation at scale.

We spent months, before rolling out guardrails or policies, creating cross-functional alignment. Data, Architecture, HR, Security, Strategy, Transformation: all in the room together, working through the questions that feel basic but aren't. What does AI-enabled actually mean in our context? What trade-offs are we genuinely willing to make? What beliefs have we always held about how work gets done that no longer hold? It felt painfully slow, but the payoff has been substantial. When your core team shares genuine mental alignment, not just vocabulary, they become force multipliers. They answer questions consistently, spot misalignments early, and model what they are asking of others.

What this requires is not a workshop. It is the willingness to surface and resolve the conflicting beliefs that most leadership teams have allowed to linger. That is harder and slower than building a platform. But it is also the only thing that makes a platform land.

✦ KEY LEARNING

Shared mental models are not a precondition for transformation. They are the transformation. Everything else is implementation.

Belief 2:

Design for adoption, not deployment (human factor as the real constraint)

Every organizational pressure in an AI transformation points in the same direction: deploy faster, show more, demonstrate progress. Deployment numbers are visible. Adoption is not. That asymmetry is where most transformations quietly fail, not with a dramatic collapse, but with a slow erosion of trust as people encounter tools they were not ready for, in workflows that were not redesigned to absorb them, solving problems they were not asked about.

The hardest call in a transformation like this, and the most important, is choosing to slow or halt deployment when adoption cannot follow. Not because the technology is not ready but because the organization is not. That decision runs against every incentive structure in a large company. It requires acknowledging that speed of deployment is not a proxy for progress, and that a rolled-out capability nobody uses is not an asset; it is a liability that makes the next rollout harder.

What this belief actually demands is a different definition of done. Not "is the technology available?" but "are people using it, trusting it, and working differently because of it?" That means involving users before redesigning workflows around AI, measuring real behavioral change and not just feature availability, and being willing to hear that something you built is not landing, and acting on that rather than explaining it away.

✦ KEY LEARNING

Deployment is an engineering milestone. Adoption is the transformation. Confusing the two is the most expensive mistake in this space.

Belief 3:

Servant leadership scales, control doesn't (enabling spirit over structure)

When you stand up a transformation unit, every organizational signal pushes you toward control: a strong mandate, central ownership, visible authority. It feels like the responsible thing to do given the stakes. It is also the fastest way to build something that nobody outside your team actually adopts. The organizations that treat AI transformation as a centrally controlled program consistently produce the same result: sophisticated solutions deployed into organizations that were never genuinely ready for them, by teams that confused rollout with change.

We designed our transformation unit around enablement rather than control. Not because we lacked ambition about the transformation, but because we understood that sustainable change at 150-team scale cannot be mandated from the center; it has to be owned locally to stick. That meant building guardrails that acknowledged uncertainty rather than pretending we had all the answers. It meant federating decisions to where the local context actually lived. It meant letting teams pull support when they needed it rather than pushing frameworks when we thought they should want them. The hardest part was resisting the pressure, internal and external, to demonstrate impact through the size of the program rather than the quality of what landed.

What this requires is a specific kind of leadership: the willingness to enable others to do more. That is not passivity. It is the only model that scales in a domain moving too fast for any central team to stay ahead of.

✦ KEY LEARNING

The transformation unit that accumulates the most authority will deploy the most. The one that distributes authority most effectively will transform the most. They are not the same thing.

Belief 4:

Partner for candor, not comfort (partners who tell hard truths)

Most consulting relationships follow an implicit script: the client defines the problem, the consultant delivers the solution, and both parties protect their respective expertise as a boundary. That model made sense when the knowledge lived clearly on one side. It breaks down entirely in a domain moving this fast, where nobody has the complete picture and the most valuable insights emerge from combining what each party knows into something neither could have reached alone.

Our approach with Nova Ventura was different. They ran the ACE Ventura experiment inside their own organization: a real product, a real codebase, real failure modes. We brought the enterprise context: 150 teams, a federated model, the organizational complexity that a controlled experiment cannot surface. The patterns in this paper emerged from that combination. That is not a typical client-consultant dynamic; it is co-authorship of a learning journey.

The belief this points to is not simply "be more open with your consultants." It is about choosing a working model where the boundary between what you know and what they know is deliberately permeable, where you build on each other's learnings in real time rather than exchanging polished outputs at defined milestones. That requires a different kind of relationship contract: less vendor management, more shared ownership of what you are trying to figure out together.

✦ KEY LEARNING

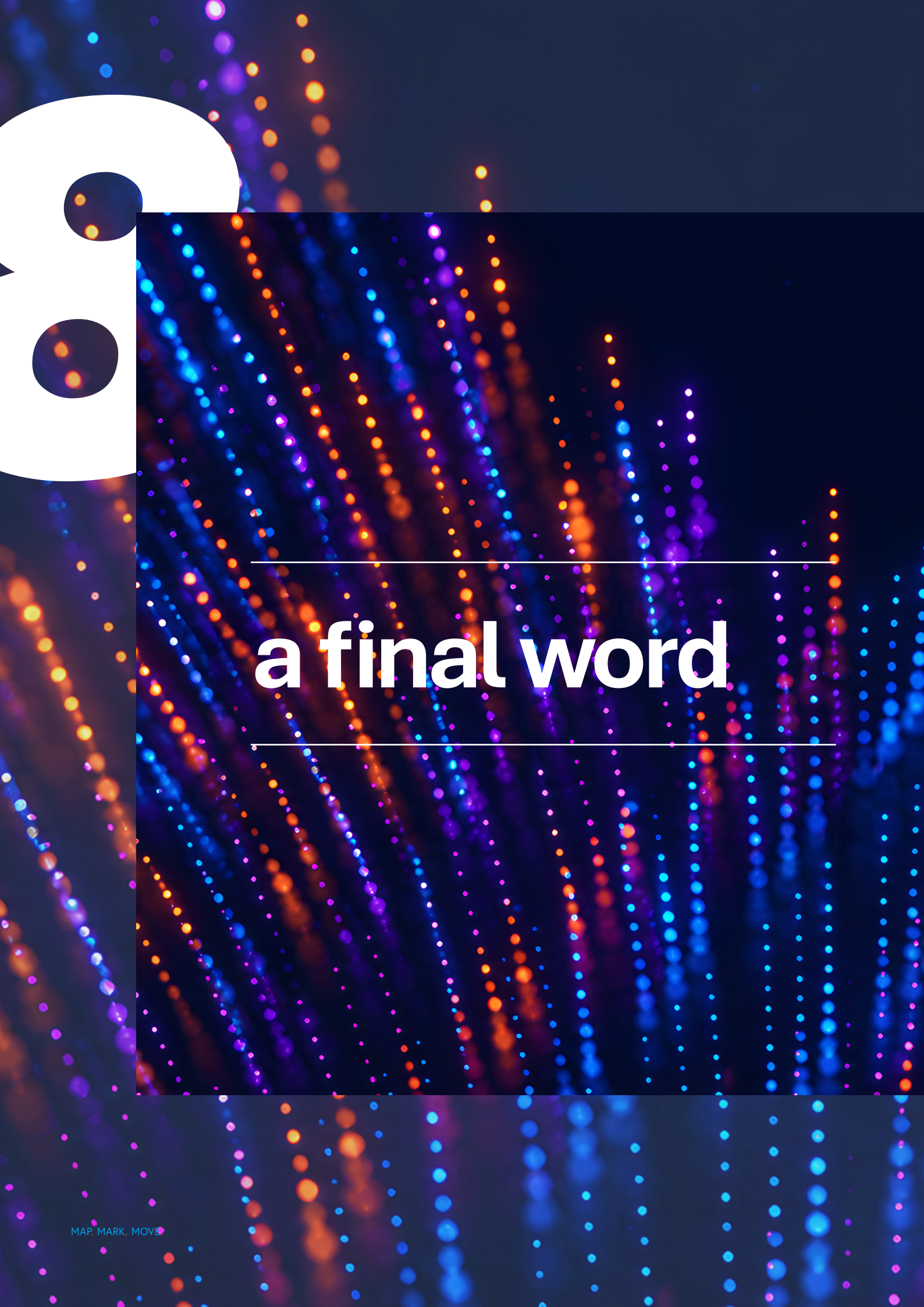
Find partners who bring something you genuinely cannot replicate internally, and then work with them in a way that permeates the boundary between your knowledge and theirs. That is where the real value lives.

7

✦ KEY LEARNINGS AT A GLANCE

- ✦ Agentic delivery does not create misalignment or unclear ownership. It removes the slack that allowed both to remain invisible. What was manageable ambiguity at low speed becomes an immediate blocker at high speed.
- ✦ Tacit knowledge is not lost in agentic delivery; it simply never enters the system in the first place. If it is not encoded, it does not exist.
- ✦ Shared mental models are not a precondition for transformation. They are the transformation. Everything else is implementation.

- Deployment is an engineering milestone. Adoption is the transformation. Confusing the two is the most expensive mistake in this space.
- The transformation unit that accumulates the most authority will deploy the most. The one that distributes authority most effectively will transform the most. They are not the same thing.
- Find partners who bring something you genuinely cannot replicate internally, and then work with them in a way that permeates the boundary between your knowledge and theirs. That is where the real value lives.



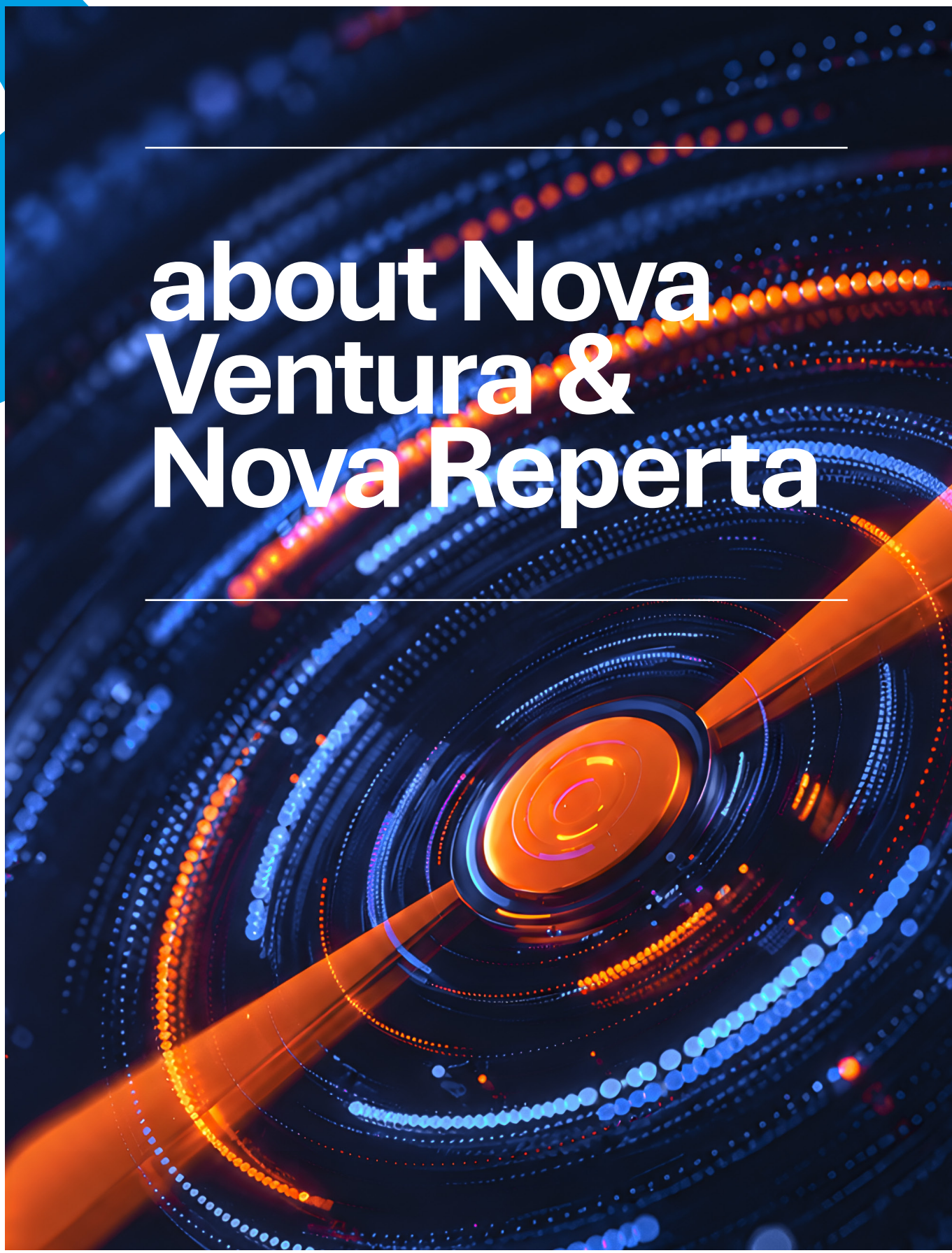
a final word

We started this as an experiment in a controlled environment, with a small team, a focused problem, and the freedom to fail without consequence. What it revealed was not primarily technical. The real discoveries were organizational: what breaks first when execution accelerates, what has to be true before AI can scale, and what kind of leadership creates the conditions for transformation to stick rather than stall.

None of this is finished. The space will keep moving, the patterns will keep evolving, and the honest answer to most questions about where this ends up is still "we don't know." What we do know is that the organizations navigating this well are not the ones with the most advanced models or the largest transformation programs. They are the ones that took the organizational prerequisites seriously before the technical ones, and kept the discipline to do that even when every pressure pointed the other way.

That is a leadership choice, made repeatedly, under pressure.
It is the only thing in this entire journey that cannot be automated.

about Nova Ventura & Nova Reperta



We are Nova Reperta and Nova Ventura: two sister firms helping organizations turn ambition into transformation that actually lands.

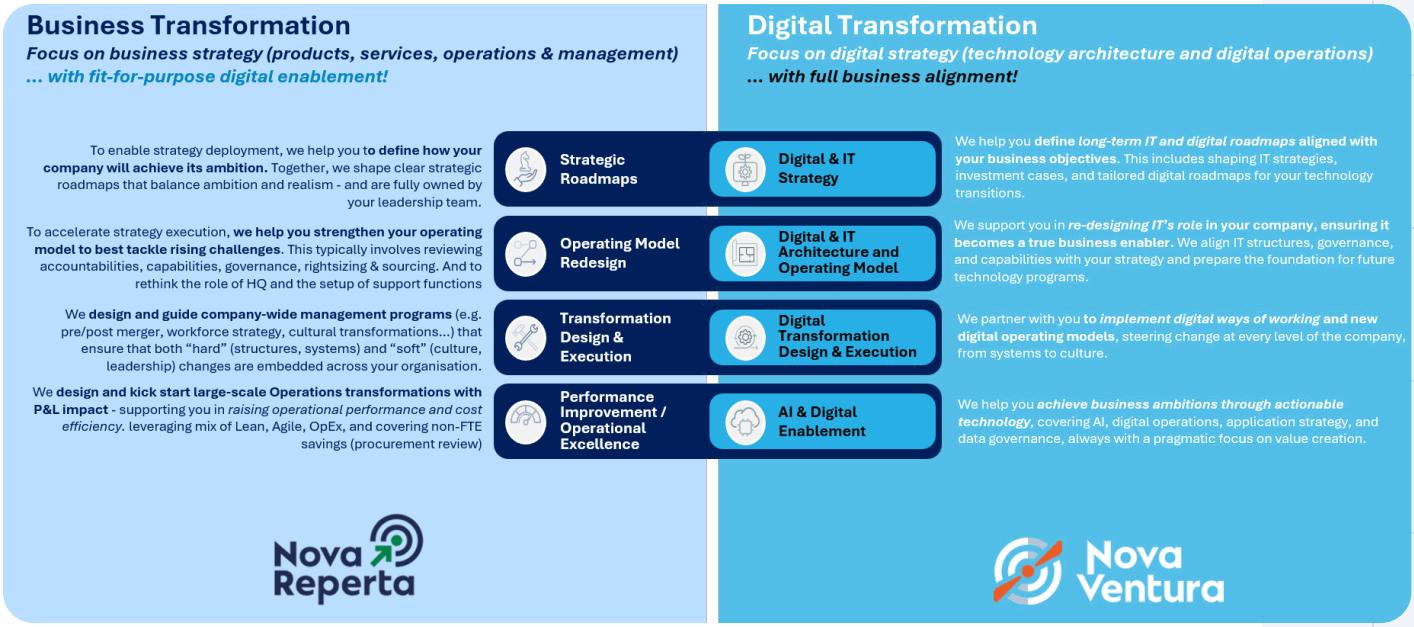
In this whitepaper, we shared what happens when AI moves from experimentation into real work. Our conclusion is clear: technology alone does not create value. Value appears when strategy, operating model, knowledge, technology and people are designed to move together. That is where we work best.

Through Nova Reperta, we help organizations clarify direction, improve performance, strengthen ways of working and make change structurally sound. Through Nova Ventura, we extend that capability into the digital domain, helping organizations align digital strategy, architecture, data, AI, technology decisions and operating models with business ambition.

Together, we cover the full transformation spectrum: people, process, data and technology. We bring business transformation and digital transformation together, not as separate tracks, but as one integrated movement from strategic intent to executable, scalable and lasting change.

Our way of working is pragmatic, senior, candid and impact-driven. We do not stop at recommendations. We stay involved to help decisions become direction, direction become delivery, and delivery become capability.

That is also why we ran the ACE Ventura experiment ourselves. We believe the best advice comes from doing the work, learning from the friction, and turning that learning into guidance clients can actually use.



appendix

10.1 BUILDING BLOCKS FOR AI TRANSFORMATION: A PRACTITIONER REFERENCE

The following framework distils the core building blocks discussed throughout this paper into a practitioner reference. These are not sequential steps. They reinforce each other and typically need to evolve in parallel. Organizations that treat them as a checklist to complete in order will struggle. Organizations that treat them as a system to develop together will move faster with more control.

1. Strategic focus

AI delivers lasting value when it is tied to business-critical capabilities, not when it is introduced as a generic productivity layer. The starting question is not "where can we use AI?" but "where does AI genuinely strengthen what matters most to this business?" Without that clarity, AI initiatives multiply without compounding.

2. Concrete starting scenario

AI becomes real when it is anchored in a scenario that is both measurable and meaningful: something that sits in the core of a value stream and solves a problem people feel every week. A strong starting scenario creates momentum and generates the early evidence needed to sustain organizational commitment through the harder phases that follow.

3. Fast learning loops

Agentic delivery changes the economics of software development: building becomes cheap, which makes it easy to drift. Fast learning loops matter more as a result, not less: early user feedback, measurable checkpoints, and deliberate decisions to stop or adjust scope before the wrong thing is built at speed.

4. Capability to build and run AI systems

AI does not scale through tooling alone. It scales when organizations build the internal capability to deliver, govern, and operate AI-enabled systems as part of daily work. That means model configurability, operational visibility into usage and cost, and guardrails that keep behavior predictable, built and owned internally, not delegated to a vendor.

5. Strong data and knowledge foundations

AI quality depends entirely on what it is allowed to use as context. Uncontrolled ingestion increases noise, contradictions, and outdated context, and erodes trust faster than almost anything else. Data and knowledge foundations need to be treated as products: intentional, governed, and continuously maintained. Curation is not overhead. It is quality control.

6. Operational discipline

AI-assisted delivery increases throughput, but does not remove, the need for engineering fundamentals. Quality standards must be encoded into pipelines and tests, not carried in people's heads. Automated CI/CD and testing are foundational from day one because agentic workflows generate large changes quickly, and manual review alone does not scale. Architecture boundaries, integration rhythms, and explicit conventions matter more when agents are involved, not less..

7. Adoption and operating model

A platform only creates value if it becomes part of daily work and stays there. Building the technology is one challenge. Making it stick is the bigger one. AI touches many parts of the organization simultaneously (decision flows, roles, governance, knowledge practices, delivery speed), which means adoption cannot be treated as a rollout phase at the end. It is the moment where AI initiatives either create lasting operational value or fade out after the first wave of enthusiasm.

The leadership beliefs and organizational prerequisites that determine whether adoption succeeds are explored in depth in section 7.3 of this paper.

Appendix

10.2 ABOUT THE AUTHORS



Benjamin Geens
Director Digital Transformation & Strategy Nova Ventura
Contact: benjamin.geens@novaventura.com
LinkedIn: www.linkedin.com/in/benjamingeens



Barbara Arnst
VP Transformation Telenet group
Contact: info@barbara-arnst.com
LinkedIn: www.linkedin.com/in/barbara-arnst-a422b8

Nova Ventura
MAP. MARK. MOVE.

BELGIUM
Koningsstraat 47,
1000 Brussels

THE NETHERLANDS
Mr. Treublaan 7,
1097 DP Amsterdam

www.novaventura.com
www.novareperta.com