THE COMPOSABLE ADVISOR COCKPIT

A Reference Blueprint
for Swiss Banking and Insurance.

adesso

CONTENT

Executive Summary Page 4

1. Why advisor productivity is the 2026 frontier in Swiss banking and insurance Page 5

2. Why these programs fail: a dual diagnosis Page 6

3. The reference architecture Page 7

4. The agentic delivery model Page 9

5. Outcomes and how to start Page 11

6. Conclusion Page 12

EXECUTIVE SUMMARY

In Swiss banking and insurance, the advisor is expensive to employ and hard to scale at the same time. Across the advisor environments adesso has assessed in Swiss banking and insurance, a relationship manager or insurance advisor typically moves between five and ten systems that do not talk to one another to serve a single client. Preparing for a meeting takes anywhere from half an hour to an hour; writing it up afterwards takes another half hour. None of this appears on a P&L as its own line, yet it shows up everywhere else – in how many clients an advisor can carry well, in how long each decision takes, and in how little time is left for the conversations that actually win business.

The market has settled on two answers, and both have run out of road. Buying an integrated vendor suite delivers breadth fast, but it ties the institution to the vendor's data model and, in time, to the same advisor experience every competitor on that suite also runs. Rebuilding the portal from scratch delivers ownership, but it takes years and carries the failure rate BCG measured in 2024: more than two-thirds of large technology programmes come in late, over budget, or short of scope.[1]

Both failures trace back to the same two roots. The first is architectural: each approach treats the cockpit as one product, when it is really a collection of capabilities that behave very differently. The second is about how the work is delivered: traditional teams add governance on top as the programme grows, instead of building it into the way the software is made. Solve one root and ignore the other, and the programme still fails – it just fails sooner.

This paper describes a way to address both roots together. The architecture is **composable**: five layers joined by stable contracts, so each capability can change on its own schedule without dragging the rest along with it. The delivery model is **agentic** – adesso's **adSCAILE** – where AI agents do the implementation while a small senior team sets the requirements, the architecture and the guardrails, and the build cycle drops from weeks to days. One half decides what to build; the other decides how to build it without ending up in the failure statistics.

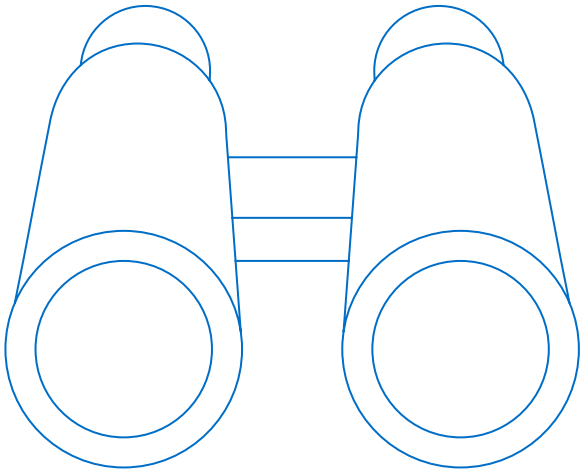
The reference engagement behind this paper is a composable advisor cockpit for a large Swiss B2C insurer, still in development. Its stated business goal is to give advisors back up to 30% of their time, a figure the programme is built to reach rather than a result already measured. The delivery model it relies on is not experimental: adesso has run **adSCAILE** in production since 2025 across several industries.
How to read this paper: a **COO** or **Head of Distribution** will find what they need in this summary and Chapters 1, 2 and 5. A **CIO** should add Chapters 3 and 4. An **enterprise architect** will want the whole thing.

[1] Boston Consulting Group, Most Large-Scale Tech Programs Fail—Here's How to Succeed, 13 November 2024, https://www.bcg.com/publications/2024/most-large-scale-tech-programs-fail-how-to-succeed.

1. Why advisor productivity is the 2026 frontier in swiss banking and insurance

Three shifts are landing at once in 2026, and together they push advisor productivity to the top of the agenda for Swiss banks and insurers. Each has been building for years; this is the year they converge.

Begin with the client. Synpulse's 2026 bancassurance research found that in developed markets nearly 70% of customer engagement now starts online, and more than half of all engagements move back and forth between online and offline channels.[2] That omni-channel pattern is now the norm for the median client, not the mark of a digital-first minority. By the time a client reaches an advisor, they have usually done their own research and narrowed the options, and they expect the advisor to already know where they are and what they are weighing. This is the wrong job for a CRM built to log conversations the advisor started. What the advisor needs is a single view that gathers up everything that happened across those channels before the meeting.



Margins are the second pressure. Swiss retail banks are caught between thin interest margins and the rising cost of regulation: FINMA, FIDLEG, revFADP. Insurers face softening fees and heavier claims. Neither can hold the cost-income ratio on revenue alone, so the savings have to come from distribution, where the advisor is the largest cost. Freeing up 30% of an advisor's time is not a rounding error. Over a five-year horizon it can be what decides whether the cost base works at all.

The third shift is AI itself, which has moved from pilot to production faster than most roadmaps assumed. Gartner expects 40% of enterprise applications to include task-specific AI agents by the end of 2026, up from under 5% today.[3] On the engineering side, McKinsey's 2026 work finds the best-performing firms gaining 16 to 30% in productivity and 31 to 45% in software quality from AI-native development; in financial services, firms running dedicated "AI agent factories" report productivity gains of 40 to 70% on greenfield builds.[4] For a CIO, putting an AI co-pilot in front of advisors is no longer the hard decision. The hard part is building an architecture beneath it that can survive the next several rounds of model and tooling change.

Put the three together and the old toolset starts to look expensive. Juggling five to ten systems worked well enough when the client walked into a branch and the advisor set the pace. It does not hold up when the client arrives already informed and judges the relationship on whether the advisor can suggest the right next step in the first minute of the call. **The real question is how to rebuild the cockpit without repeating the failures of the last decade of attempt.**

[2] Jason Amayun, The Future of Bancassurance: Unlocking Data, Digital, and Omnichannel Experiences, Synpulse, 20 January 2026, https://www.synpulse.com/en/insights/the-future-of-bancassurance-unlocking-data-digital-and-omnichannel-experiences. Figures derive from Synpulse's mystery-shopping research in developed markets.

[3] Gartner, Gartner Predicts 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, Up from Less Than 5% in 2025, press release, Stamford CT, 26 August 2025, https://www.gartner.com/en/newsroom/press-releases/2025-08-26-gartner-predicts-40-percent-of-enterprise-apps-will-feature-task-specific-ai-agents-by-2026-up-from-less-than-5-percent-in-2025.

[4] Charlotte Relyea and Martin Harrysson, The AI Revolution in Software Development, McKinsey & Company, April 2026, https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/the-ai-revolution-in-software-development. Top-quintile findings draw on McKinsey's analysis of c. 300 publicly traded companies; the financial-services figure refers to a documented enterprise agent-factory engagement.

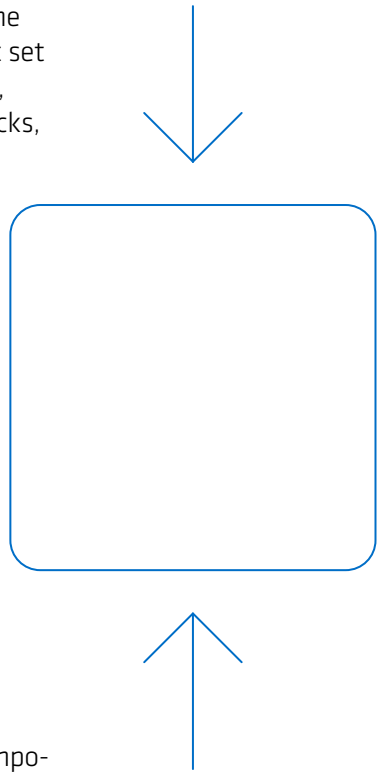
2. Why these programs fail: a dual diagnosis

Over the last decade, two approaches have dominated. Both are still being rolled out in Switzerland today, and both have hit a wall.

The first is the **integrated suite**: Avaloq, Temenos or Finnova in banking, Guidewire or Sapiens in insurance, Salesforce Financial Services Cloud or Microsoft Dynamics across both. The promise is tidy – one data model, one upgrade path, one vendor to hold accountable – and the integration it delivers is real. For an institution without the engineering muscle to build, it is often the sensible call. The limits show up later. Anything the institution wants that does not fit the vendor's data model becomes a customisation, and every customisation makes the next upgrade harder. The institution does not control when or how the suite evolves. And because rivals run the same suite, the advisor experience tends to converge: whatever once set one bank apart from another stops living in the cockpit and moves to the brand, the people and the products.

The second is the **custom rebuild**: a new portal on a modern stack, built in-house or with an integrator. Ownership is the draw, and on paper it is a strong one. The track record is where it falls down. BCG's 2024 study of more than a thousand executives found that over two-thirds of large programmes miss on time, budget or scope, and traced the failures to three things: governance added on top of delivery, no reusable patterns to build on, and a siloed data model that quietly reassembles itself inside every new monolith.[5] A rebuild also forces every capability onto a single release schedule and every workflow onto a single data model. More often than not it ends as a second monolith – new stack, same shape – after a year or two and several million francs.

Both approaches rest on the same assumption: that the cockpit is one product evolving at one pace. In reality it is a bundle of capabilities that move at very different speeds. The parts that set an institution apart, such as the next-best-action engine or the way a meeting gets prepared, need to change every few weeks. The commodity parts, like e-signing, video and identity checks, are best bought and left alone. The regulated parts, such as FIDLEG disclosures, FATCA and CRS reporting, AML alerts and claims audit trails, move on a slow and controlled cycle measured in quarters. No single architecture serves all three well. The painful trade-offs of the last decade were never built into the problem; they came from forcing this mix into one rigid structure.

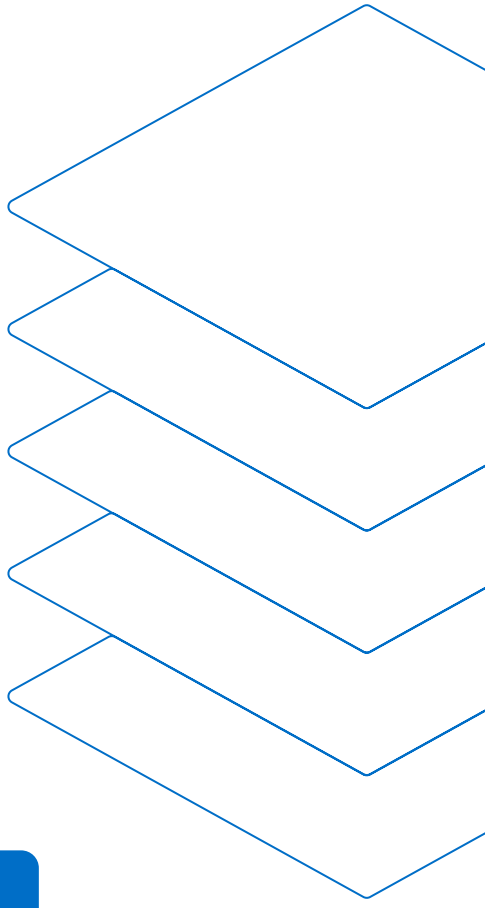


Architecture, though, is only half the diagnosis. BCG's third cause – governance that piles up as the programme matures – is really about how the work is delivered, not how the system is designed. Hand a composable architecture to a traditional time-and-materials team and it takes on the same debt, just spread across more pieces; it fails in a different way, but no less often. Three problems recur whatever the architecture looks like. Coordination starts to eat the programme: the larger it grows, the more effort goes into keeping teams in sync rather than building anything. Nothing compounds: the eleventh capability costs about what the first did, because the patterns from earlier work are never captured and reused. And governance arrives as a string of manual gates, until releases that should ship weekly slow to once a quarter.

So the fix has to work on both fronts at once, because the two problems feed each other. Composition handles the architectural problem; agentic delivery handles the coordination, leverage and governance problems. Either one on its own falls short. A composable architecture built the traditional way still drowns in coordination. Agentic delivery aimed at a monolith just builds the monolith faster. Together, they are what let adesso commit to a fixed price and a four-to-six-month timeline without that being reckless. The next two chapters take each half in turn.

[5] BCG, Most Large-Scale Tech Programs Fail (2024). Survey of more than 1,000 C-suite executives across 20 sectors and 59 countries.

3. The reference architecture



What follows is a **reference architecture**, which is a different thing from a system design. It sets out what each part is responsible for and how the parts talk to each other, without naming products. Two institutions that follow it will share the same layers and the same contracts between them, while sensibly choosing different technology underneath according to what they already run and what their teams know well: Angular or React, Java or .NET, open-source or managed identity. The layers and their contracts stay constant from one institution to the next; the technology beneath them is a local choice.

There are five layers. Each has a clear responsibility, a set of reference patterns, and its own rate of change, which need not match its neighbours'. What separates one layer from the next is a contract rather than a shared implementation, and that is what allows different teams to own different parts of the cockpit without the whole thing collapsing back into a monolith.

Layer	Primary responsibility	Reference patterns	Cadence
Composition shell	Host and route independently deployed micro-frontends; enforce design system and accessibility.	Module Federation; Web Components; design tokens.	Slow
Identity & access	Authenticate, manage sessions, federate identities, enforce step-up on sensitive operations.	OpenID Connect, OAuth 2.1, FIDO2 passkey; open-source or managed IAM.	Slow
Context fabric	Aggregate a real-time unified client view from systems-of-record and expose it to all consumers.	Backend-for-Frontend; GraphQL federation or REST gateway with event sync.	Medium
AI co-pilot	Generate suggestions, drafts and next-best-actions against client context, with auditable guardrails.	LLM gateway; vector store for retrieval; agent orchestration.	Fast
Integration	Connect to core systems: banking core, policy admin, CRM, document, payment, reporting.	API gateway; event broker; legacy adapters.	Mixed

Table 1. The five layers of the composable advisor cockpit.

The composable advisor cockpit - reference architecture

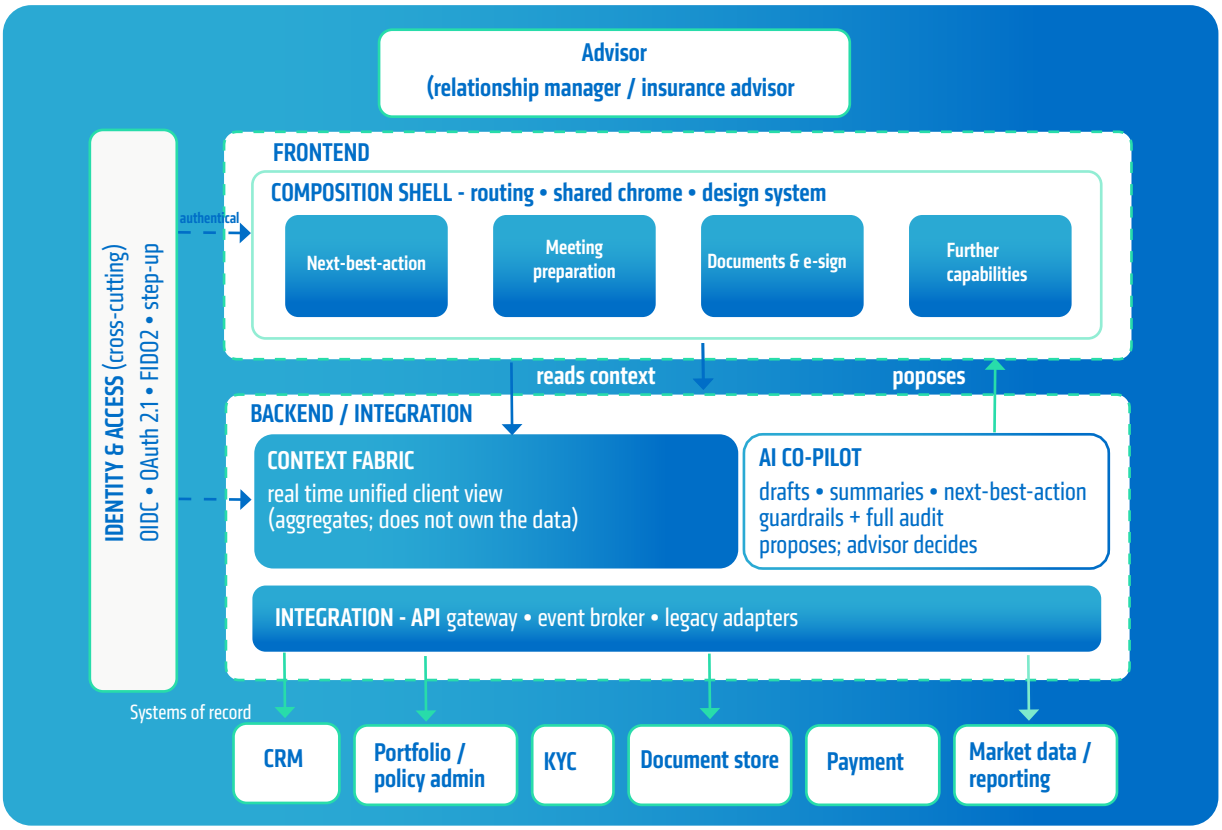


Figure 1. The composable advisor cockpit: reference architecture. Frontend and backend tiers, with identity and access as a cross-cutting concern.

Three of the five carry most of the weight. The composition shell is a thin, long-lived host: it routes between independently deployed micro-frontends, supplies the shared chrome and enforces the design system. It holds no business logic of its own. Everything the advisor actually uses lives in a micro-frontend that another team owns and ships. The context fabric is what makes the whole asymmetric problem manageable. It pulls together a live, unified view of the client from the underlying systems and serves that view to whoever needs it, while only ever aggregating: the source systems stay the system of record, and the moment the fabric starts owning data rather than assembling it, it has quietly become one more enterprise data warehouse. The same fabric that feeds the advisor cockpit can feed the customer-facing omnichannel touchpoints, which is how an institution keeps what the advisor sees aligned with what the client saw a moment earlier. The AI co-pilot then works on that unified view to draft messages, summarise situations and propose next steps, always within its guardrails and always logged. Whether model inference runs in a Swiss-resident cloud or on the institution's own infrastructure is a deployment choice the architecture keeps open; revFADP data residency and FINMA outsourcing requirements are met by design either way. One line stays firm: the co-pilot proposes, the advisor decides. Anything covered by FIDLEG suitability or AML stays the advisor's call. The co-pilot assembles the context and prepares the paper trail, but it never stands in for the advisor's signature.

Five contracts hold the layers together, and each one has to be stated rather than assumed. Runtime federation lets each capability deploy on its own schedule instead of waiting for a shared release. Identity travels across the whole surface as short-lived, narrowly scoped tokens; when an advisor steps up to a sensitive action, that elevated state rides along as a claim and then lapses, rather than unlocking the entire session. The context broker gives every part of the cockpit one stable way to ask what is true about a client, and absorbs any change in the source systems behind that single interface. The design system is run as a product in its own right: teams take it as a dependency and cannot quietly override it in their own corner. And every capability ships through its own pipeline, with quality and compliance checks built into that pipeline rather than into a shared release train. That last discipline is the one thing that keeps a composable system from sliding into a distributed monolith.

Designing today for tomorrow's generative UI
Today the AI co-pilot sits beside the interface: it occupies one part of a screen that people designed and built. The next step changes that relationship. In what is being called generative UI, the agent assembles the interface itself, at the moment it is needed, from a library of approved components. In April 2026 Google published A2UI v0.9, an open specification that lets an agent render its own interface using whatever component library the host already has, with reference support for the main frontend frameworks.[6] It is too early to build on this – the standard is still at version 0.9 – but four decisions taken now make it an upgrade later rather than another rebuild: keep the component library as a versioned, first-class asset; publish the design system as machine-readable tokens, not just a style guide; describe the UI as data rather than hard-coded screens; and make sure the shell can render a component tree handed to it at runtime, not only the micro-frontends registered in advance. Get these right and, when generative UI is ready, it slots in as one more renderer behind contracts the institution already has.

4. The agentic delivery model

This chapter is about how the architecture actually gets built and shipped. In adesso's experience, this is the part that separates a composable programme that works from one that quietly turns into a distributed monolith.

One clarification first, because this paper talks about AI agents in two different roles. The AI co-pilot from Chapter 3 lives inside the finished cockpit and helps the advisor day to day. The delivery agents in this chapter work at build time: they write, test and document the cockpit, then have nothing to do with how it runs. For the rest of this chapter, "agents" means the second kind.

Most of the talk about AI in software engineering lumps together four quite different levels of involvement. They are not points on a smooth line. Moving from one to the next is a step change, not a small improvement.

Level	Name	Pattern	Where humans
01	AI-assisted	Reactive AI invoked on demand for autocomplete or chat.	Developer is the sole actor
02	AI-enabled	AI integrated into discrete SDLC sub-steps: testing, documentation, review.	Developer retains full control
03	AI-driven	Agents operate within defined loops on larger task sets.	Humans orchestrate, AI delivers
04	AI-native	Agent-based AI is the foundation; small teams, ultra-short cycles, autonomous implementation.	Humans think as architects

Table 2. Four levels of AI involvement in software engineering. The 03-to-04 shift enables fixed-price commitments on complex builds.

[6]Google, A2UI v0.9: A Protocol for Generative UI, Google Developers Blog, 17 April 2026, https://developers.googleblog.com/a2ui-v0-9-generative-ui/. Open, framework-agnostic specification with reference renderers for Flutter, Lit, Angular and React; standards at this maturity evolve rapidly and the version should be re-checked before publication.

Most engineering organisations in Swiss financial services sit at Level 01 or 02 today. A few are piloting Level 03 on greenfield code. Very few have reached Level 04 at any scale, because Level 04 is not mainly a tooling change. It asks for a different operating model: smaller senior teams, work fed in as clear specifications, and cycles measured in days. **adSCAILE**, the adesso Smart Cycle for AI-enhanced Lightweight Software Engineering, is adesso's way of running Level 04 as a repeatable process.[7]

The core idea is to pull apart two things that traditional delivery runs together. Deciding what to build, how it should be structured and what a good result looks like stays with people. Doing the building – the code, the tests, the documentation, the refactoring – is handed to agents that work in long loops under human review. A few principles follow from that split. Sprints give way to something faster, because an agent loop closes in two to four days rather than two weeks. The work becomes continuous rather than batched: the agents keep going overnight, and the team picks them up and redirects them each morning. Agents are specialised by job – analysis, implementation, testing, review, documentation – with clean handovers between them. And people stay in charge throughout, setting the boundaries and correcting course while the agents build inside them.

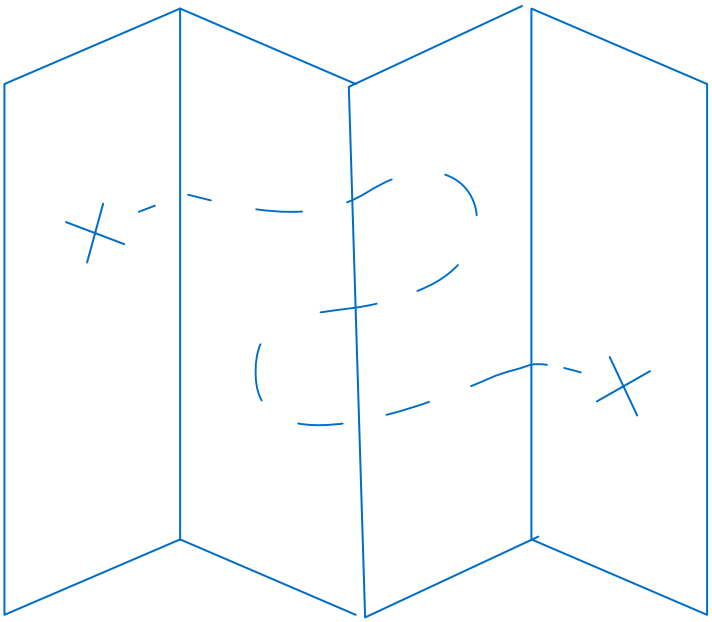
Run capability by capability, the cycle moves through the same five steps each time: people set the requirements and the architecture, people and agents together write the component specification, agents do the implementation, review combines automated checks with human judgement, and operation carries on after release. That cycle maps onto the five layers, and it takes the three problems from Chapter 2 apart one by one. Small teams working from clear specifications cut out most of the coordination that swallows a traditional eight-team programme. Patterns from one capability are captured and fed back in, so the eleventh capability costs less than the first instead of the same – the compounding that traditional delivery never manages. And the compliance and security checks run inside the build loop, with the documentation agent producing the regulatory artefacts as it goes, so governance is part of the work rather than a layer added at the end. That is how a regulated Swiss institution can ship at the speed the architecture allows instead of the speed the audit committee tolerates.

This is not theory. **adSCAILE** has been running in production at adesso since 2025. In one publicly documented case in the utilities sector, a conventional Scrum team and an **adSCAILE** agentic team worked in parallel on the same live system, each shipping twice a week; the agentic team rewrote the front end from Angular to React without a single line of hand-written code, drawing on several thousand pages of domain documentation held as retrieval context.[8] That case is in utilities rather than financial services, and the distinction matters. The cockpit blueprint is the same method adapted to a regulated setting, where the specifications the agents work from carry FIDLEG, FINMA outsourcing rules and revFADP data-residency constraints. The method travels across industries; the specifications do not.

None of this means the work runs itself. The architectural decisions stay with people: which composition pattern to use, where the layer boundaries fall, how tightly the co-pilot is fenced in. Those play out over years and cannot be handed to a cycle that turns over in days. Regulatory accountability stays with people too. The hardest shift, though, is cultural. The organisation has to stop prizing engineers for how much code they write and start prizing them for how well they specify and review, and that shift takes far longer than rolling out the tools. It is the most common reason a promising pilot never scales.

[7] adesso SE, adSCAILE – Agentic Software Development, company website, accessed May 2026. Source for the four-level maturity model, the four design principles, and the einfachnetz.de production engagement (TransnetBW & VNBs Baden-Württemberg, utilities sector).
[8] adesso SE, adSCAILE (see note 7). The einfachnetz.de engagement was delivered for TransnetBW and the distribution system operators of Baden-Württemberg; delivery metrics and C-level statements are reported in the same source.

5. Outcomes and how to start

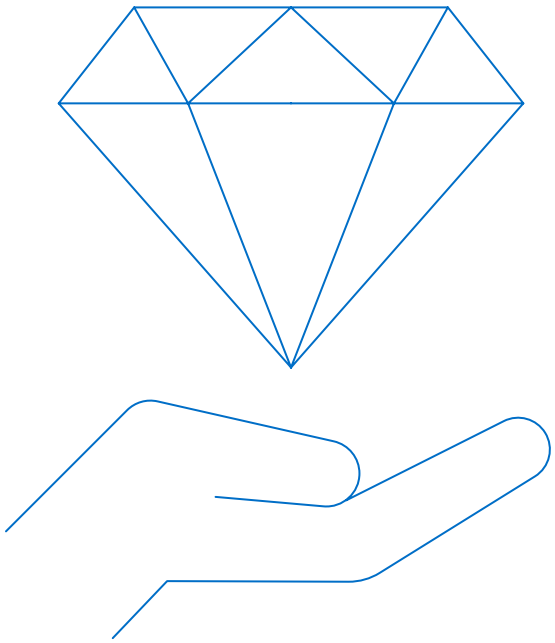


The reference engagement behind this paper, a large Swiss B2C insurer, is still in development. Its stated business goal is **to give advisors back 30% of their time.** That number is an estimate the programme is built to reach, not a result already measured, and it is worth being clear about which it is. What carries over to other institutions is the mechanism, not the figure: when advisors stop moving between systems and stop re-keying data by hand, that time comes back for advising. The delivery promises – a first capability in production within four to six months, at a fixed price – come from the operating model in **Chapter 4**, not from any market average. They hold as long as the institution can supply clear requirements and people empowered to decide, which is itself something the engagement is set up to secure.

Rather than quoting a bigger number, it is more useful to triangulate from three reference points of a different nature. The market benchmark: McKinsey's figures cited in **Chapter 1** put top-quintile gains from AI-native development at 16 to 30% in productivity, and at 40 to 70% for financial-services agent factories on greenfield builds. The production evidence: the utilities engagement described in **Chapter 4**, where the delivery model sustained two releases a week on a live system with no hand-written code. And the programme goal itself: 30% of advisor time, set deliberately within what the first two points suggest is reachable. An institution's own result will depend less on the technology than on two things it controls: how many of the advisor's daily systems the first capabilities actually replace, and how quickly its people decide when decisions are put in front of them.

None of this requires switching off the existing portal first. The approach is a strangler pattern: the composition shell goes up alongside the current environment, and capabilities move across one at a time, most valuable first, each shipped as its own micro-frontend behind the shared identity and context contracts. The old portal shrinks capability by capability rather than being replaced in one risky cutover, which keeps the transition far less dangerous than a full rebuild.

A first engagement is kept small on purpose. It starts with a journey audit: one current advisor journey mapped end to end, with real numbers, to set the baseline and pick the first capability to tackle. From there comes the reference-architecture decision, fixing the layer contracts and the stack choices for this particular institution. Then one capability goes into production through the **adSCAILE** cycle, which proves the architecture and the delivery model on real ground rather than in a slide deck. Alongside it, the engagement stands up the operating model – the small senior teams, the design authority, the governance built into the pipeline – so the second and third capabilities can follow each quarter. The goal of the first engagement is not a finished cockpit. It is one capability live in production, and enough confidence in the organisation to scale from there.



6. Conclusion

In Swiss banking and insurance, advisor productivity is the contest of 2026, and **the cockpit is where it is won or lost**. The cockpit is also the advisor-facing surface of the wider omnichannel platform, which is why getting it right pays off well beyond the advisor's own desk. The two familiar answers have stalled for the same reason: both treat the cockpit as a single product, and both are built by teams that take on more governance debt the larger they grow. Getting past that does not mean doing either one better. It means putting two moves together: a composable architecture that lets each capability move at its own pace, and an agentic delivery model that makes that architecture something an institution can actually ship, on time and at a fixed price.

The real decision is not which technology to choose. It is whether **to treat the advisor cockpit as a strategic asset** at all. Treated as one, it shapes whether the advisor experience sets the institution apart or drifts down to match whatever every competitor on the same suite offers, whether the institution owns its roadmap or rents it from a vendor, and whether the next wave of AI lands as an upgrade or as one more rebuild. The institutions that get there first, pairing the right architecture with the right way of building it, are the ones that will **turn advisor productivity from a line of cost into a genuine advantage.**

We look forward to speaking with you.

Gianluca Colaianni
Head of Omnichannel Platforms
Competence Centre

gianluca.colaianni@adesso.ch

adesso Schweiz AG
Vulkanstrasse 106
CH-8048 Zurich
T +41 58 520 97 10
E marketing@adesso.ch
www.adesso.ch

adesso