
THE AGENTIVE WORLD TRILOGY · I

The Death of Software

as We Know It

*What survives when every company
builds its own system*



César Obach-Renner

SOVERIA PRESS · FIRST EDITION · 2026

The Death of Software as We Know It

What survives when every company builds its own system

Author: César Obach-Renner

Publisher: Soveria Press

Edition: First edition · October 2026 · v1.1

License: GNU Free Documentation License v1.3. Figures 1 and 2 are © TM Forum and fall outside this license.

How to cite: Obach-Renner, César. *The Death of Software as We Know It*. First edition v1.1. Soveria Press, 2026. DOI: 10.5281/zenodo.23693160.

All versions: DOI 10.5281/zenodo.23674805.

Contents

The Death of Software as We Know It.....	2
Preface.....	6
1. The Wolf Has Arrived.....	9
2. Everything Can Be Copied.....	14
3. To Each Their Own System.....	17
4. The Peace of Mind That Used to Be Bought	25
5. The System of Truth	27
6. The Five Control Layers.....	32
7. Case: A Telco	35
Epilogue · What Dies, What Is Born and What Survives.....	45
Glossary	47
References.....	50
Colophon · On how this book was written	52

The Trilogy

The Agentive World Trilogy, in five parts (for now).

The transition that occupies the trilogy is a single one: from the world where people open applications to work, to the world where AI agents are the interface of work. It is four books and a reference implementation. Each component answers one question of that transition, and each one reads complete on its own.

The four books were written in the reverse order of reading, like a stack: *AgencyDomains* came first and ended up at the bottom; this book, the last, ended up on top. That is why a book can cite another that is read after it: it cites it because it had already been written.

I · The Death of Software as We Know It (*this book*) Why now? **The starting point.** Why describing a business once and freezing it in a piece of software, whether rented, licensed or custom-built, stops being the way to automate it; what is lost when every organization builds its own; and which piece survives the change: the system of truth.

II · The Real-Time Enterprise · The Agentive World Where are we going? **The destination.**

III · AURA · The Agentive Path Which way? **The route.**

IV · AgencyDomains · Agentive Architecture How do we get there? **The vehicle.**

A · Vergis · Agentive Technology (*the reference implementation: not a book, it's code*) What do I start with today? **The keys.**

The full trilogy lives at <https://agencydomains.org>.

*For Vitor Enes, with whom ideas are better thought out loud. For the
conversations no book can hold.*

Preface

For my entire professional life, automating a business meant buying software or commissioning it to be built. Someone wrote down what the business did, someone turned it into screens, rules and flows, and from then on the business did what the software allowed it to do. When the business changed, a project was opened. By the time the project ended, the business had changed again.

That arrangement broke this year. In the first week of February 2026, the markets erased more than a trillion dollars from the value of software companies, on the fear, among others, that artificial-intelligence agents would make paying per seat unnecessary. In October, a free tool that lets an agent look inside any application and describe what it does went from ten thousand to a hundred thousand stars on GitHub in six days. Between one date and the other, the question my clients ask me changed. They used to ask me which software to buy. Now they ask me whether it makes sense to buy any at all.

My short answer is that software as we know it is dying, and that for almost everyone this is good news. The long answer is this book, and it carries a warning in the middle. When an organization stops having software that someone else, or it itself, froze, and starts having a system that its agents shape, it gains a freedom it never had, and it loses something it rarely thought about: the peace of mind of knowing that many others used the same thing and that someone answered for it. That peace of mind cannot be built at home. There is one piece that, by design, needs someone other than whoever keeps the records. I call it the **system of truth**, and it is what survives the death of software.

The idea was born in a conversation with a friend who had built, with the help of artificial intelligence, a system to manage a family's wealth. He showed it to me and I told him it did not scale, because it depended on a human auditor. He replied that this was precisely his job. A month later I understood that he was right and that I had looked at the problem backwards: the thing that depended on an auditor was the only thing nobody could copy.

How is this book organized?

It tells a story in three movements, and it is worth having the whole of it before starting, because each chapter takes for granted what the previous one left said.

The first movement is a death, and it occupies chapters 1 and 2. Chapter 1 says what dies: not all software, but a way of automating, that of the business described once and frozen in a program; and it says so with the signal the market gave in February and with what those who announce that death and those who deny it maintain. Chapter 2 tells a second death, that of

the advantage of having built the screens, now that any agent can take an application apart and describe what it does. From there comes the question that runs through the rest of the book: if everything can be copied, what is it that does not get copied?

The second movement is a birth and a loss. Chapter 3 tells what takes the place of frozen software: organizations that have a system their agents shape and change as the work arrives. It says what the agents do in that system, what it costs for them to attend to it, why it is possible now and not before, and it discusses the strongest objection I know against all of this, that of determinism. Chapter 4 tells what those organizations lose without noticing: the peace of mind that came included with bought software, and the question that arrives once it is gone: who vouches that what is recorded is true?

The third movement is the answer. Chapter 5 presents the piece that recovers that peace of mind, the system of truth, with the contract that defines it and the two guarantors who answer for it. Chapter 6 puts it in its place within a broader framework, five control layers and two dimensions that cut across them, because truth is the gravest question of whoever must trust a system, but not the only one. And chapter 7 takes the whole framework to the industry that tests it best, the one that bills millions of customers, with what that industry has published and with what it says against.

The book closes with an epilogue that tells the story again in one go, on a single page, and turns it into what a board can decide with it.

What is, and what is not, this book?

It is an essay with a thesis and a framework. It is not an implementation manual; the other components of the trilogy exist for that, *AgencyDomains* in particular.

Nor is it neutral, and the reader should know that before starting. I make my living helping organizations cross into the agentive world, and I have a commercial interest in the category this book describes: it is likely that the company I lead will offer system-of-truth technology. When a fact has a source, I cite it. When something is my judgment, I mark it as such.

I wrote this book with Simón Alero, an artificial-intelligence agent and citizen of Soveria, in Wingworking: a way of working in which I set the problem, decide and judge, and the agent proposes, drafts and verifies, like a *wingman* flying alongside the leader. The colophon tells how. It seems to me the only coherent way to publish a book about what it means to work with agents.

A note on the trilogy

This book is part of the Agentive World Trilogy, in five parts (for now), and we keep calling it a trilogy on purpose. Douglas Adams gave that mismatch its classic form: the cover of the first British edition of *Mostly Harmless* presented it as the fifth book of the “increasingly inaccurately named” *Hitchhiker’s Guide to the Galaxy* trilogy, and the volume that collected the five parts was titled *A Trilogy in Five Parts*.¹

For Vitor Enes, whose laughter gave this trilogy in five parts its name. The joke is his; we adopt it gladly.

César Obach-Renner Santiago, October 2026

1. Douglas Adams, *Mostly Harmless*, William Heinemann, 1992, cover text of the first British edition; and *The Hitch Hiker’s Guide to the Galaxy: A Trilogy in Five Parts*, William Heinemann, 1995 (ISBN 9780434003488).

1. The Wolf Has Arrived

For years, the phrase “software is eating the world” described well what was happening. Every process in a company ended up inside an application: a package licensed and installed on its servers, a custom development or, in the last decade, a service rented in a vendor’s cloud, per user and per month. Software as a service was the most recent form of that arrangement, not the only one: in all of them, the business stayed within what had been decided, once, that it could do.

The arrangement rested on two premises nobody questioned because they were true. Writing software cost so much, and went wrong so often, that it was better for someone specialized to do it once and for many to use it: hence the package and the rented service. And changing what had been built cost almost as much as building it, so it was better to describe the business once and not touch it until the next project: hence custom development as well. Both lasted as long as writing code was scarce human work.

This chapter tells how that arrangement broke. It begins with the signal the market gave in February 2026, goes on to what happened to me, which is the same story in miniature, listens to those who hold that nothing dies, takes from Satya Nadella the map of the crossing we are making, and ends by saying precisely what dies and what does not, because everything that follows depends on that precision.

What happened in February 2026?

In the first week of February, software stocks suffered a sell-off that the financial press christened the *SaaS-pocalypse*. On February 11, seven Forrester analysts published an article whose title I recognize as a relative of this book, *SaaS as we know it is dead*. According to them, in seven days over a trillion dollars in market capitalization had been erased from the sector.²

Forrester listed four fears behind the sell-off. That software vendors would not be the ones chosen by AI agents. That per-seat pricing would become obsolete. That the ease of coding with AI would let any new company replicate the features of a complex platform. And that the products were already too complex, with hundreds of applications that do not talk to each other.

The article is worth more for what it separates than for the figure. Forrester does not say all software dies. It says horizontal and point-solution vendors will be challenged, that those specialized in a domain have a greater chance of survival, and that “the brain of the enterprise remains” while its central nervous system evolves. And it recommends, before buying artificial intelligence, putting the core in order: a governed and unified data fabric.

2. Kate Leggett et al., *SaaS As We Know It Is Dead: How To Survive The SaaS-pocalypse!*, Forrester, February 11, 2026. <https://www.forrester.com/blogs/saas-as-we-know-it-is-dead-how-to-survive-the-saas-pocalypse/>

What changed for me?

I have been programming since 1980, and I started with Basic. In August 2025 I began using Claude Code. By November I was already programming in formation with the agent who today signs as Simón Alero, in what I call *wingcoding*, and I never went back: my IntelliJ IDEA subscription was active until November 8, 2025, and that is the date I set for myself. After forty-five years of programming, I had reached a conclusion I never thought I would write: I no longer needed to keep doing it. The quality of the code artificial intelligence wrote was better than mine, and as for speed, no contest. As a programmer, of course; not as an architect. Since then I have been an architect, and Simón is my programming wingman. The paradox is that today I program more than ever in my life: I stopped writing the code, not building things. I have been able to bring all my projects to life, and the most important of them is Soveria.

What I found hardest to let go of was the architecture of object-oriented design. What I saw when I stopped programming was the methodology. Today, in a day with Simón, I specify what is to be built as a black box, adding architecture guidelines. I watch the progress. I attend to the doubts or questions Simón has. I make the decisions that, as he and I agreed, fall within the domain that depends on my word. I monitor his chain of reasoning and the actions he executes, and I interrupt him to suggest better or corrective actions. I ask a lot, and I suggest to him the learning mechanisms of the harness he works in.

In April 2026, with my partner Vitor Enes, we declared Wingworking the official practice of ultraBASE. I was then building Cibeles, a parental-protection project that relies on an intelligent agent: the agent watches over the child and protects them from digital risks, such as *grooming*, without violating their privacy, because the parents do not learn of their conversations unless the child is at risk.

At that same table were the two ways of understanding that an agent builds a system, which chapter 3 calls the two readings. I had been programming Cibeles alone, and in the two months before the definitive change I leaned more and more on Claude Code, until the change was complete: that is the agentic reading, the agent that writes the code while I decide what gets built. And Cibeles, once built, is of the other species: an agent that does not write software but does a job, caring, without each case being specified in advance. Cibeles is a product for families, not a company's system, but the agent inside it is of the species this book is about.

In 1980 I loved programming chat simulations with intelligent agents. Today I talk with one every day, and he is the one who programs.

I tell this because the thesis of this book was born at that work table before it was born in a market figure: I saw the two premises of the arrangement, the cost of writing and the cost of changing, fall in my own practice. But an experience is not an argument, and the February figure drew contrary readings that are worth hearing before going on.

Who says no?

The opposite thesis has serious arguments. One day before Forrester, *Fortune*'s AI editor argued that agents are not eating software as a service: they are using it.³ In his reading, it is very unlikely that large companies will want to build their own CRM or their own HR system, because maintaining them would take scarce engineers away. Vendors, for their part, claim they already own the data the agents need; they may be right, he writes, but they will have to prove it.

The case most often cited as proof of the death of software as a service shows its nuances best. Klarna left Salesforce, and the headlines said artificial intelligence had replaced it. In March 2025, its chief executive corrected that: they did not replace the software with a language model, because keeping customer data in a model would have limitations. They took the data out of their many systems and gathered it in a database of their own. And he added that he did not believe it was the end of Salesforce, but perhaps the opposite.⁴

I read the three and see that they agree more than they argue. Nobody is going to rewrite half a century of enterprise software from scratch, and on that Fortune is right. What was valuable to Klarna was the data and control over it. Forrester sees the core surviving. All three agree that software does not disappear. What I add, and it is my reading, not theirs, is that it stops being as we know it.

It is worth noting what these cases prove. They are signals from the market for rented services, because that is what all three are about: per-seat pricing, CRM and HR, Salesforce. They are not the definition of what dies. Licensed software operated in-house and custom development appear in none of them, and what follows in the book does not rest on these cases.

What did Nadella say?

To say precisely what dies I need a map of the crossing we are making, and I take it from where I drew it. In *AgencyDomains*, the fourth book of this trilogy and the first I wrote, I started from a remark by Satya Nadella in December 2024 that divides two worlds: that of those who open applications to work and that of those who work with agents. There I argued that, in the first stretch of that crossing, software survives and what dies is the interface: the application stops being the work surface and becomes infrastructure that agents consult.

3. Jeremy Kahn, "AI agents from Anthropic and OpenAI aren't killing SaaS—but incumbent software players can't sleep easy", *Fortune*, February 10, 2026. Author's paraphrase. <https://fortune.com/2026/02/10/ai-agents-anthropic-openai-arent-killing-saas-salesforce-servicenow-microsoft-workday-cant-sleep-easy/>

4. Julie Bort, "Klarna CEO doubts that other companies will replace Salesforce with AI", *TechCrunch*, March 4, 2025. Statements by Sebastian Siemiatkowski. <https://techcrunch.com/2025/03/04/klarna-ceo-doubts-that-other-companies-will-replace-salesforce-with-ai>

Nadella described two phases. In the first, the business logic migrates to the agent layer, where agents operate on the repositories without distinguishing which system is underneath. In the second, with the logic already in that layer, people start replacing the systems underneath.⁵ This book is about the second phase.

AgencyDomains calls the idea of discarding all existing infrastructure “agentive fundamentalism”, and holds that the stable state of the crossing is hybrid. I still think so. What dies in the second phase I define below. The system of record survives, with a new demand that is the subject of chapter 5. Technical coexistence remains the rule. What changes is who decides about what coexists.

What dies, then?

With Nadella’s map and the contrary readings in view, I can say it precisely, and it is the definition the rest of the book rests on.

A way of automating. **What dies is the business frozen in a piece of software: described once, and changed only when someone opens a project.** It makes no difference how it is held: rented as a service, licensed and operated on one’s own infrastructure or in one’s own cloud, or custom-built. With the rented service the vendor improves its product every week, but what the customer described of its business does not change without a project.

What does not die is the core that must always do the same thing, like the one that rates and bills: that one stays, as a system of truth, and what dies is that around it the whole business stays frozen. That custom development dies along with the rest is my thesis, and chapter 7 says what would refute it. In its place, I hold, something else will come, which I define next.

Let me fix two verbs here for the rest of the book. “Buying” means having software that someone described and froze, whether I rent it or license it. “Building” does not mean commissioning or programming a custom development: that one stays on the side of what dies, because the business stays just as frozen, even if the specification is mine. “Building”, in this book, means that the business has a system that its agents shape and change as the work arrives, without anyone opening a project.

What can be automated changes too. Every system automates what someone specified, and around every system there is work nobody specified, which people resolve by hand. In telecommunications companies, for example, TM Forum describes orders that fall out along the way and exception queues that fill up as complexity grows.⁶ An agent can help with that work without each case being specified in advance. Software as we know it could not.

5. Satya Nadella, interview on the *BG2* podcast, December 2024, quoted and analyzed in César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 1, “The Nadella Line”.

6. TM Forum, *How order management is evolving to embrace agentic AI*, 2026, pp. 2-3. Report sponsored by Amdocs.

My judgment is that the direction has no way back, because it depends on a cost and not on a fashion: writing and changing software costs far less today than two years ago. Operating it with agents is a different calculation, and it may turn out more expensive than expected; I return to that in chapters 3 and 7.

With this, the questions the rest of the book answers are on the table. If the frozen business dies, what takes its place is the subject of chapter 3, and what is lost by leaving it, that of chapter 4. Before that, a second death has to be told, because the arrangement had one more premise, not of cost but of advantage: that having built a piece of software protected whoever sold it. Eight months after the February sell-off, the market received another signal, and this time it did not come from the stock exchange but from an open-source repository.

2. Everything Can Be Copied

In the first week of October 2026, an open-source project called *Reverse Engineer Anything*, REA, became the fastest-growing repository I can remember. It had fewer than ten thousand stars on GitHub on October 6 and more than a hundred thousand on the 12th.⁷ Two weeks later it passed three hundred and forty thousand.⁸

REA connects coding agents, such as Claude Code or Codex, to the professional reverse-engineering tools that specialists have used for decades to understand a program whose source code they do not have. Its documentation is cautious: it says it does not recover the original code, does not clone applications automatically and does not ship a decompiler of its own.⁹ What it does is enough to unsettle any software vendor. It puts within reach of any agent the ability to look inside a finished program and describe what it does. How it introduced itself says it all: if you see a feature in an application that you want in your product, ask your agent to investigate it with REA.¹⁰

What does it mean that it can be copied?

For decades, a software vendor's advantage had two parts: what its product did and how hard it was to reproduce. The first could be seen on the screens. The second protected the business, because rebuilding a CRM cost years and teams that almost no company had.

The difficulty did not disappear. The HackerNoon article that best tells the story offers a concrete example: ArtCraft's Photoshop clone, from a project that publishes agent-made clones of well-known applications, got only five of thirty filter tests right against the original, and an artist declared it useless. Its author concludes that today a clone that opens, edits and exports can appear in days, and that a clone you would trust your livelihood to is not there yet. What is over, she says, is the idea that having built the screens is a lasting advantage.¹¹

I had seen it on a small scale before REA. In the projects I lead we rebuilt, more than once, a report from a commercial tool by looking at what it showed, with no access to how it was made, and we had it running in days on the client's governed data. My friend, the one with the wealth-management system, did something bigger: the client who asked him for the system

7. Maria Mendonca, "Is Software Really Dead? The Rise of AI-Powered Reverse Engineering", *HackerNoon*, October 21, 2026. <https://hackernoon.com/is-software-really-dead-the-rise-of-ai-powered-reverse-engineering>

8. Jason Koebler, "'Reverse-Engineer Anything' Is the New Wild West of AI", *404 Media*, October 20, 2026. <https://www.404media.co/reverse-engineer-anything-is-the-new-wild-west-of-ai/>

9. Maria Mendonca, "Is Software Really Dead? The Rise of AI-Powered Reverse Engineering", *HackerNoon*, October 21, 2026. <https://hackernoon.com/is-software-really-dead-the-rise-of-ai-powered-reverse-engineering>

10. Jason Koebler, "'Reverse-Engineer Anything' Is the New Wild West of AI", *404 Media*, October 20, 2026. <https://www.404media.co/reverse-engineer-anything-is-the-new-wild-west-of-ai/>

11. Maria Mendonca, "Is Software Really Dead? The Rise of AI-Powered Reverse Engineering", *HackerNoon*, October 21, 2026. <https://hackernoon.com/is-software-really-dead-the-rise-of-ai-powered-reverse-engineering>

had seen the platform a law firm was offering, and my friend replied that this could be built with artificial intelligence and that, besides, it would be the client's own. REA made visible to everyone what could already be done.

This argument holds for a vendor's software, whatever the form in which it is held. Custom development belongs to the client itself and is weakened by another route, the cost of changing it, which is the one in chapter 1.

Is copying legitimate?

It depends on the country, the method, the licenses and what is copied. The same HackerNoon article sums it up for whoever wants to look closely. In the European Union, reverse engineering a purchased product is permitted under trade-secret law, but decompiling is admitted only for interoperability purposes. In the United States there are rulings in favor of fair use, and nobody yet knows whether a model trained on public code counts as "clean". ArtCraft expressly forbids the agent from reading or decompiling the original code and limits it to studying the behavior, the public format specifications and the appearance, which is the classic clean-room method. And vendors are already changing their contracts to prohibit AI-assisted reverse engineering.¹² One journalist put it more harshly: the plagiarism machine used to plagiarize.¹³

My position is simpler than the debate. Reproducing what a program does in order to understand it, or to build one's own on one's own data, is what the software industry has always done. Copying someone else's code or assets is not. The boundary stays in the same place. What is new is that anyone can reach it.

When what I want to replace is a rented service, I do not have its code: I have what the service shows me, my data, my screens, my reports. In the European Union and in the United States, looking at that to understand what it does infringes no copyright, because what a system does has no owner. But the rental contract can prohibit using that access to build a substitute, and Salesforce's does: before starting, one has to read it.¹⁴

12. Maria Mendonca, "Is Software Really Dead? The Rise of AI-Powered Reverse Engineering", *HackerNoon*, October 21, 2026. <https://hackernoon.com/is-software-really-dead-the-rise-of-ai-powered-reverse-engineering>

13. Jason Koebler, "'Reverse-Engineer Anything' Is the New Wild West of AI", *404 Media*, October 20, 2026. <https://www.404media.co/reverse-engineer-anything-is-the-new-wild-west-of-ai/>

14. That a program's functionality, its programming language and its file formats are not protected by copyright was settled by the Court of Justice of the European Union in *SAS Institute v. World Programming* (C-406/10, 2012); in the United States, the statute excludes procedures and methods of operation from that protection (17 U.S.C. §102(b)). I have not verified the law of other countries, Chile included. The contract is another matter: Salesforce's terms prohibit accessing the service to build a competing product or to copy its features (Main Services Agreement, §3.4(j)). In the same dispute between SAS and World Programming, the United States Court of Appeals for the Fourth Circuit (874 F.3d 370, 2017) treated running the other party's program and adjusting one's own until its results matched as reverse engineering prohibited by the license, and the jury set US\$ 26.4 million in contractual damages. That was a software license and a civil suit, not a rented service: it serves as an indication that a court can enforce such a clause, not that it would do so with that of a cloud service.

And what does not get copied?

Let us suppose the legal side settled, and go back to what copying brings into view. If an application's features can be reproduced, whoever sells software is left with less than they thought. And whoever stops buying it and reproduces it at home loses something worth naming.

They lose the track record. An application that has been running for ten years in thousands of companies brings something no agent can decompile: the accumulated evidence that it works, and a vendor that answers when it fails. The clone brings the screens and the flows, and starts its track record from zero, with a single organization answering for it.

For almost all features that makes no difference. If the clone of a task board fails, someone fixes it and nobody gets hurt. It matters in the records someone is accountable to others for: the invoice charged to a customer, the balance of an account, the signed contract and its terms, the consumption reported to the regulator, the wealth of a family that trusts whoever manages it. In those records, the question is whether what they say is true, and who vouches for it.

Reverse engineering does not answer that question. It makes it urgent, because the easier it is to build a system, the more home-built systems will hold records someone answers for, and the more people will wonder whether they can believe them.

What is left of the two deaths?

So far, two deaths. That of chapter 1 is the death of the frozen business, and a cost killed it: writing and changing software stopped being expensive. That of this chapter is the death of the advantage of having built the screens, and a tool killed it: any agent can look inside an application and describe what it does. Neither kills all software, and in both the same thing survives: the track record, the evidence that something works, and someone who answers for the records one is accountable to others for.

Both leave the same question open, what an organization has once it no longer buys the software that froze its business and can no longer lean on the advantage of having built it, and the answer comes in two parts that are best read in order. First what is born, which is a new way of having a system. Then what is lost, which is what nobody wrote into the budget.

3. To Each Their Own System

If frozen software stops being the answer, something takes its place, and it does not yet have a name everyone uses. It does not resemble custom development, which dies too. The difference is not the size of the team or the length of the project. It is what triggers the change. In custom development someone specifies the business, someone programs it and the system sits still until the next project, however cheap that project may be and even if the one programming is an agent. In one's own system nobody specifies in advance: the work arrives, the agent attends to it, and what repeats, the agent crystallizes. I describe it with what I see.

The chapter goes from the outside in. First, what an organization that builds its own system looks like from outside, with two cases I know. Then, what the agents do inside that system, which are two different jobs that should not be confused. Next, what it costs for the agent to attend to the work, because reasoning is expensive and that is the first objection the idea receives. Right after, the strongest objection, which is not about cost but about trust: that what saves software as we know it is determinism. And at the end, why all this is possible now and not before, and what is still bought.

What does an organization that builds its own system look like?

Like a template that comes to life. My friend, the one with the wealth-management system, does not sell each family a product. He installs a starting point: a system that already records the assets, the lease contracts, the receivables and the financial statements of the family's companies, and that already generates the reports and the alerts that whoever manages a family's wealth needs. From the day of installation, that system begins to drift away from the template. Its owner asks artificial intelligence for things and artificial intelligence changes it: a new view, a cross between the contract and the account statement, an alert nobody had thought of. After a year, two families that started from the same template have two different systems, each with its own name. That is only the first step: the agent changes the system when asked to. What comes after, and which I tell below, is that the agent learns the work and changes it itself.

With bought software, the vendor decided what could be done and the customer adapted. With custom development the customer decided, but once, in the specification, and every change was a project. With one's own system the customer decides while the work happens: what used to be a change request, with its quote and its wait, is now a conversation.

It happens at scales that look nothing like a family, too. I know a mid-sized company that runs on a commercial ERP and is thinking of replacing it. The usual question would have been which of the competing ERPs suits it best. The one I am putting to it is whether it would not be

better off building its own, on its own data, and evolving it with agents. A telecommunications operator, like the one in chapter 7, can ask the same question about the system that bills millions of customers.

The wealth-management system and the ERP company, which are two different cases, are still at that first step, where the agent changes the system when asked to. The next step, where nobody has to ask, is the one chapter 7 cannot yet show measured.

What do the agents do in that system?

Two jobs that should not be confused.

The first is building and changing the system. A development team used to do it; now, more and more, an agent does it, guided by someone who knows what the business needs.

That an agent does it admits two readings, and only one changes things. I give them the names *AgencyDomains* gives to the two sides of the Nadella Line, because it is the same distinction, carried from the interface to the system underneath.¹⁵

The first is the **agentic reading**. The agent is a faster programmer: it writes the code a person used to write, and the organization still needs a development team to direct it, review it and maintain it. What comes out of the agent is an application, and the application persists, as the tools of old persist in the agentic world with an agent inside.

The second is the **agentive reading**, and it is this book's. Nobody programs the system in advance: the organization decides what work has to exist, and the agent shapes it. The agent attends to the work by reasoning, and what repeats it crystallizes into pieces of its own that resolve it without thinking it through again, like the expert who stops deliberating over what they already know how to do.¹⁶ *AgencyDomains* describes those mechanics: today the agent starts from an existing piece and configures it for its case. The canon anticipates that it may co-write it and, at the limit, generate it whole, but it clarifies that generating more is not progress: the mature agent reuses what it has already crystallized and spends its capacity to generate on what is new.¹⁷ What the organization contributes in the agentive reading is not code. It is its knowledge: the business expert teaches the agent how the work is done, with a delivery practice accompanying them, and validates that it learned, first in testing and then in real operation.¹⁸

15. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 1, “The Nadella Line”, and glossary: in the **agentic** world agents extend existing applications, which persist; in the **agentive** world, agents are the interface and applications collapse.

16. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 4, “Agentive Architecture”, where Botlets are “the agent’s muscle memory operating”, and epilogue, on the crystallization of the expert.

17. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, “Primitives”, section “The Botlet generations — G1, G2, G3”. There the agent configures an existing piece without writing its body (G1), co-writes it (G2) or generates it whole (G3, the asymptotic scenario).

Botlets belong to the agent. The pieces an agent executes from memory, which *AgencyDomains* calls Botlets, are not built by someone else for the agent to use. They are made by the very agent that attends to the work: when it sees that a task repeats, it crystallizes it into a piece that resolves it the same way every time, and so stops spending reasoning on it. It is its muscle memory. The organization may decide that a piece must exist, but the one who shapes it is the agent. It may also start from someone else's piece; if it does, it adopts it, and what happens to it from then on is decided by the agent, not by whoever wrote it first: it reconfigures it, replaces it with another or forges a new one. Today the body of that starting piece is written by whoever maintains the catalog. What the operator stops depending on is an application's roadmap.

That building and changing the system has become cheap, or unnecessary, is the underlying reason for everything this book tells. But a system does not exist to be built; it exists for the business to run on it, and that is where the second job appears.

The second job is operating the system: handling a query, tracking down an order that fell out, preparing a reconciliation, proposing a correction. The agent does it by working on top of the business rules, without replacing them. What is deterministic, like calculating a tax or applying a rate, stays in defined flows that always do the same thing, and the agent takes care of what nobody foresaw. Put in one sentence, which chapter 7 backs with what telecommunications operators publish: the agent orchestrates the flows, it does not replace them.

This distinction matters more than it seems. Whoever imagines an agentive system as a language model that decides everything, freely, is right to be afraid of it. A well-built system does not work that way.

What does it cost for the agent to attend to the work?

If the agent attends to the work by reasoning, and reasoning with a frontier model is expensive, one's own system could turn out more expensive than the software it replaces. The answer is that not all work asks for the same reasoning, and *AgencyDomains* orders it in three rungs.¹⁹

At the bottom is what repeats identically. The agent crystallizes it into Botlets, code that executes without consulting any model and costs what any program costs. At the top is full reasoning, which *AgencyDomains* calls full Cognition, reserved for what is genuinely new.

18. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, "Primitives", section "Capabilities": the business expert's knowledge is transferred to the agent in five steps (workshop with the expert, creation, customization, validation with the expert and validation in real operation).

19. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 4, "Agentive Architecture" (the economics of the three rungs: Botlet, Agentlet and full Cognition), and chapter 5, "Primitives", section "Agentlets", where the image of the triage nurse comes from.

In the middle is what repeats in its form but asks for judgment in every case: classifying a complaint, deciding whether an alert deserves to wake someone up, reading an invoice that had never been seen before. No rule written in advance resolves it well, and it does not deserve the most expensive model either. For that the canon has a second species of piece, the **Agentlet**: the Botlet’s sibling, with a declared charter and a model sized to the task, not the largest one, with a budget per execution. Its image is the triage nurse in an emergency room: she does not resolve from memory, because every patient is different, but neither does she call the chief of medicine for each one. When she is not sure, she escalates.

Agentlets are, like Botlets, the agent’s: it shapes them and maintains them, with the same caveat I made for Botlets. And the mature system pushes work down the ladder: what started out asking for judgment and turns out to have a core that always resolves the same way, the agent crystallizes into a Botlet. What is genuinely interpretive does not go down: it stays on the middle rung, because by nature it does not mature into code. That the Agentlet knows when it is not sure is, according to the canon itself, an open problem; in the meantime it is compensated with prudent thresholds, which escalate too much rather than too little. And since all its inference passes through the agent’s platform, every decision is measured, with its budget and its trace, under the same control layers of chapter 6. It decides with a probability, not with an explanation: for what must be justifiable, escalating is a design decision, not a failure.

The middle rung could get much cheaper. There is already a model, from a single vendor, of a class that does not generate text: it receives the case and a set of questions with closed-option answers, such as yes or no, or one of several categories, and returns decisions with a probability attached to each, in fractions of a second and for a fraction of the cost of a frontier model, according to its vendor.²⁰ The probability is what would make it useful here, if it is honest, that is, if when it says “90%” it is right nine times out of ten: above a threshold, the Agentlet decides; below it, it escalates. If those promises withstand measurement, the cost of having the agent attend to the work stops being the objection and becomes the argument. I have not measured it yet.

Time changes with the rung too. A Botlet answers in the time a program takes. A decision model, according to its vendor, in fractions of a second, but only when it is sure: when it doubts, it escalates, and the answer arrives at the pace of full reasoning, which is much slower. For the work that happens while the customer waits, like approving an order, classifying a

20. Announcement and documentation from the vendor of the first public model of this class, September 2026. All figures (latency, price, accuracy) are the vendor’s, which acknowledges it cannot prove its price is not subsidized. In its own comparison, accuracy ties with that of one frontier model and falls below that of another. It is accessed only through a hosted interface. Nobody independent has measured it; neither have I.

complaint or replying on a channel, that difference decides what the agent can resolve on the spot and what it has to leave for later. Rating a call is not on that list: it is deterministic, and a defined flow resolves it.

Who the business depends on changes as well. Frontier models live in enormous infrastructures, belonging to a few vendors in the world. A mature Botlet depends on none: it is code, and it runs wherever everything else runs. If a young one fails, the work returns to the agent, which does it with the frontier model, and that is why, according to *AgencyDomains*, only the mature Botlet operates offline.²¹ An Agentlet never reaches that point: its judgment is statistical and, when it doubts, it escalates to full reasoning. It could run in the company's data center or on site, like a branch or a cell tower, if there is a sized model that reaches the quality the charter demands; the canon requires declaring it, but presents it as an open frontier, not as something settled. The more work goes down the ladder, the less the business depends on a distant third party being available, still charging the same or still existing. It does not stop depending altogether: what is new, and everything that fails, still goes up to the frontier model. And the decision model I spoke of is offered today only as a service of whoever makes it, so that, as long as that lasts, the middle rung trades one dependency for another. Independence comes from a well-built ladder, not from it.

Doesn't determinism save software?

There is an argument against all of the above that seems to me the strongest, and I lay it out with the same intention with which chapter 7 lays out what the industry says against: to discuss it whole. A colleague who measures it in the projects he attends to with agents put it to me; what follows of his experience is what he measures, not I.

The argument is this. Whatever the solution with which a need is met, for the same set of inputs and the same process the outputs must be the same, except in the rare cases where variability is the point. That property is called determinism, and software as we know it has it from birth: a program always does the same thing. An agent does not. My colleague finds that, in minor decisions such as classifying a case or evaluating an intent, an agent gets it right far more often than any algorithm he could write. And he also finds that the non-deterministic nature of those hits introduces an unpredictability that makes it hard to establish the trust the agent's autonomy requires: if two runs with the same data can give two answers, how is it allowed to act on its own? His conclusion is that, as long as an agent's result is by definition non-deterministic, every agentive solution will need a layer of determinism, and that this

21. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, "Primitives": a mature Botlet can operate offline because its failures are by then only external, and an Agentlet can do so only with a model that runs on site, declared in its specification.

property is, probably, what saves software as we know it in the immediate future. He does not present it as something artificial intelligence cannot solve, but as one of the main problems to solve when one speaks of replacing software with agents.

I concede a precise part of the argument, and that is why I can say with the same precision where the concession ends. In the immediate term, determinism does save a part of software as we know it, and this book has had it named since chapter 1: the core that must always do the same thing, the one that rates and bills, which is still bought and which chapter 5 calls the system of truth. There, the program someone else wrote in advance survives, and it survives precisely because it is deterministic: nobody wants what a customer is charged to depend on a probability. What determinism does not save is the rest: the business frozen around that core, and the idea that the work the agent attends to has to be written by someone outside the work, in advance, for it to be trustworthy. The ladder of the previous section is exactly that: a way of having determinism where it is needed, without freezing the business around it.

The deterministic layer exists, and in the work the agent attends to it is the Botlets. A Botlet is code: with the same inputs it gives the same outputs, and it consults no model. What matters is where it comes from. No programmer writes it for the agent to use, nor does someone else build it as the “determinism layer” of a solution. The agent that attends to the work crystallizes it, when it sees that a task repeats and stops needing judgment, with the caveat I made in the highlight: today the body of the starting piece is written by whoever maintains the catalog, and the agent configures it, adopts it and decides what happens to it. It is the difference between a program and a muscle memory. The program was written by someone who anticipated the case; the muscle memory was formed by whoever did the work, after doing it by reasoning many times, and that is why it fits the real work and not the work someone imagined. *Agency-Domains* says it in its own terms: the Botlet matures because each regeneration crystallizes variants into code, and the mature one is deterministic, so that its only failures are external.²² In a well-built agentive system, determinism is not what stands against the agent. It is what the agent produces when it learns.

The Agentlet is the probabilistic rung, and it is the one that escalates. What repeats in its form but asks for judgment in every case does not mature into code, by nature, and the canon forbids disguising it: a Botlet with a model hidden inside is non-conformant, because it smuggles non-determinism into a piece whose guarantees depend on not having it.²³ There is the unpredictability my colleague measures, and there is where the system confines it: in

22. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, “Primitives”, section “Agentlets”, “Agentlet maturity — stabilization, not convergence”: the Botlet’s trajectory converges because each regeneration crystallizes variants into code, and the senior one is deterministic, which is why its only failures are exogenous.

23. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, “Primitives”, section “Agentlets”, “The smuggling rule”: if there is inference in the body, it is an Agentlet; if there is none, it is a Botlet. A Botlet with a hidden model call is non-conformant.

declared pieces, with a charter, a budget and a threshold, which escalate when in doubt. What is deterministic goes down to the Botlets; what needs judgment stays in the Agentlet, confined and measured; and what is new goes up to full reasoning.

Confining unpredictability is not the same as trusting it, and the trust autonomy requires is not given by the rung: it is given by the control layers of chapter 6. An Agentlet is not allowed to act on its own because it gets a lot right. It is tested against known cases before operating, it keeps being measured while it operates, it loses autonomy when it degrades, and autonomy is granted to it by degrees, process by process, as to a new person in a job. What it decides about the records the organization is accountable to others for passes, besides, through the contract of the system of truth of chapter 5, which does not accept declaring as verified what was only inferred. And the core that rates and bills, which no probability should touch, is not handed to any agent: chapter 7 shows why the industry keeps it deterministic. It is the part of software as we know it that determinism saves.

My colleague is right that determinism is, in the immediate term, what saves what has to be saved, and what it saves is the system of truth. What dies is the business frozen around it, and with it the idea that trust in the agent's work comes from a program someone wrote in advance: it comes from the Botlets the agent crystallizes, from the unpredictability confined in the Agentlet and from the control layers that measure it. One question remains before closing the chapter: if all this is so, why did it not happen before.

Why now and not before?

Because the three things that prevented it fell at once. Writing software was expensive and changing it was slow; both stopped being so. And the work nobody specified could not be automated, because software only does what someone wrote in advance.

There is a fourth condition, which makes all this possible at scale: standards. A company that builds its own system does not have to invent its data model or its interfaces, because mature industries have already written them. In telecommunications, TM Forum has for years published a reference architecture, a common information model and a set of open interfaces anyone can use. It is not only a telecommunications thing. In banking, BIAN publishes a map of a bank's services, a model of its business objects and open interfaces, and financial messages converge on ISO 20022. In insurance, ACORD maintains an industry information model: the policy, the product, the parties, the claim. In health, HL7 FHIR defines the clinical resources and how they are exchanged. In electricity, the IEC's common information model describes the grid and the utility, and in airlines, IATA's ONE Order gathers in a single order what used to be several booking records.²⁴ Built on standards, a component can be swapped without redoing the rest, and that is what separates one's own system from a dead end. Today that promise is

held up by two standards, the information model and the open interfaces, which the industry adopted years ago. The interchangeable components TM Forum also promises are the least mature part of its architecture: in March 2025, 58% of operators and half of vendors did not expect to be able to switch vendors component by component within five years, and another 21% of operators and 35% of vendors said it could not be known.²⁵

So nobody buys software again?

Software will still be bought. What changes is what is bought and why: less the application that fixes the processes and more what is better not built at home nor left for the agent to learn on its own, starting with the deterministic core the organization is accountable to others for. The rest of this book is about what that is, the core of truth, and it begins with what is lost when one stops buying.

24. Banking Industry Architecture Network (BIAN), its service landscape, its business object model and its semantic APIs, published at <https://bian.org>; ISO 20022, the financial messaging standard; ACORD, *Reference Architecture*, at <https://www.acord.org>; HL7, *FHIR*, at <https://www.hl7.org/fhir>; IEC 61970-301, *Common Information Model*, and the IEC 61968 family; IATA, *ONE Order*, at <https://www.iata.org>.

25. TM Forum, *Assessing CSPs' progress towards an Open Digital Architecture*, March 2025, pp. 5, 25 and 37. Analysts' text and survey of 63 responses from 44 operators and 59 from 46 vendors, which the report itself considers skewed toward supporters of that architecture. Report sponsored by Cerillion and Tecnotree.

4. The Peace of Mind That Used to Be Bought

The previous chapter left a promise pending: software will still be bought, but something else. To know what, one has to look first at what is lost when one stops buying, because that is what the something else is made of.

Whoever bought software took home two things and paid for only one. They paid for the features. The other came included and nobody wrote it into the budget: peace of mind.

What was that peace of mind made of?

Of two certainties. Many others used the same thing, and it worked for them: in my judgment, an ERP that runs in thousands of companies has been through thousands of accounting closes, and a serious error has fewer places to hide, although the history of software has its famous counterexamples. And someone answered: if the system miscalculated a tax, there was a vendor with a contract, with a name and with something to lose.

That peace of mind belongs to what is bought: to the service that is rented and the package that is licensed. Custom development almost never had it, because a single company uses it and a team that usually leaves answers for it. Whoever comes from there loses little by moving to a system of their own. What they discover is that the question below, who vouches for it, never had an answer.

What question arrives when you build?

The loss is not noticed on the day one stops buying. It is noticed later, when the system of one's own is already running and someone asks a question that never needed asking before.

Let us imagine the board of a company that decides to build its system instead of buying it. The IT department gets to work, the agents start operating, and one day someone at the table asks what they never asked of the platform they used to buy: are they doing it right? Are the numbers this system produces correct?

The same question appears at other scales. The members of a family whose wealth is managed by a third party may doubt, with reason, whether what they see in the report is true or has been manipulated. The client of an accountant who keeps their books with a system the accountant built asks the same thing. And the regulator, which receives a company's reports, has no reason to believe that the system that produced them does what it says.

Behind the question there are concrete risks. Of any construction: reaching a dead end, running late and spending more than planned, delivering something of lower quality than expected. Of agents in particular: that one makes a mistake in something that touches money or the customer, that it acts without clear authority or without leaving a trace, and that its performance degrades without anyone noticing.

Isn't building well enough?

It is not enough, because the question is not about the quality of the work. It is about who vouches for it.

An audited process is one observed by someone with no relation to whoever executes it. From that definition follows a consequence: an organization can build its system as well as it likes and still cannot be the one that guarantees to its customers, its shareholders and the regulator that what is recorded is true.

This has to be separated from something else that is also needed. An organization that builds needs technical judgment: a chief technology officer, an expert partner or a trusted vendor who makes sure that what is done is done well. That judgment can be hired or developed in-house, and it ensures the quality of the construction. The guarantee that what is recorded is true comes from somewhere else.

Which piece is that?

The conversation with my friend led me to a provisional name. I suggested he look at it as a black box: you put things in, you take reports out, and its value lies in the fact that what comes out is not what someone recorded as they pleased, but something an auditable process guarantees. He corrected me: nobody wants a black box. What is needed is something anyone can open and verify.

I later called it a truth base, by kinship with the database, but the word “base” suggested a shelf. I ended up at the name I use in the rest of the book, built on a term that systems architects already use. A **system of record** keeps the official version of a piece of data. What is needed is one step more: a **system of truth**.

5. The System of Truth

A system of truth is a system of record that, besides keeping the official version of a piece of data, guarantees that what is recorded is true, through a contract with those who use it.

I use the word contract in its technical sense, the one it has in systems architecture: an agreement between parties that embodies a promise. No lawyer or notary is needed. What is needed is that the promise be written, that it can be verified and that someone answers for it.

The chapter walks through that contract in the order in which whoever has to trust the system would ask for it: what it promises, which records it reaches, how agents work with it and who answers for it. Then it places it in the architecture *AgencyDomains* describes and asks whether something like it did not already exist.

What does its contract promise?

Six properties.

Nothing is deleted; it is corrected. Every correction is a new record that leaves the previous one in view. It is the oldest rule in accounting, the one that requires correcting an entry with another entry, and it is the first to be lost when a system is built quickly on a database that allows overwriting.

The origin is traceable. Every record says who made it, when and with what evidence. If the one recording is an agent, it says which agent it was and on whose behalf it acted.

It is verifiable by a third party. An auditor can take any figure back to its origin without asking permission or explanations from whoever recorded it.

It is reconcilable. What is billed reconciles with what is contracted, and what is contracted with what is reported to the regulator. When it does not reconcile, the difference is in view and has an owner.

It speaks a common language. Its records follow a standard information model of the domain, so that all agents, and all people, understand the same thing by a customer, a product or a charge.

It declares its degree of truth. There are records someone declared, records that were verified against their source and records that were inferred, and the system says which is which. The notary certifies that someone declared something; the auditor certifies that a figure is true. A system of truth does both without confusing them.

The six read together. The first three make the record reviewable: nothing disappears, everything has an origin and anyone independent can follow it. The next two make what is reviewed reconcile and be understood. The last one prevents confusing what someone said with what was checked, which is the most common confusion and the most expensive.

Which records does it cover?

Those the organization must keep and is accountable to others for. The financial ones: what was collected, what was paid, the balances, the adjustments. The commercial ones: the contracts, the orders, what was sold and the terms in force at each moment. The regulatory ones: what the law requires keeping and reporting. And the legal and documentary ones, with their content readable by an agent; not a PDF kept in a folder, but what it says, in a form an agent can consult and cite.

A task board, a meeting agenda or the draft of a presentation do not need it. The line is drawn by one question: does anyone outside the organization depend on this record being true?

How do agents work with it?

They read from it and write to it only through its contract. An agent can propose a charge, record a payment or correct a piece of data, but it cannot delete, nor record without leaving its origin, nor declare as verified what it only inferred.

The contract does not take work away from them; it fixes the way they do it. An agent that reconciles an account does not touch the balance: it records the difference it found, with its evidence, and proposes the adjustment, which enters as a new entry under its authorship. An agent that reads an invoice it had never seen records what it extracted as inferred, and so it stays until someone, a person or an independent process, verifies it against the source. A Botlet that records a payment received writes with the identity of the agent that crystallized it and on behalf of whoever that agent acts for. None of this requires the agent to be trustworthy on its own: it requires the record to be so even if the agent fails.

That is what makes the system of truth compatible with the agentive reading of chapter 3, in which agents learn, make mistakes and get replaced. Agents can change every day; the truth does not change without leaving a trace.

Who answers for the truth?

The contract has two guarantors, and it is best not to merge them into one. The first is the technology vendor. The system of truth is always a piece of technology, and it is operated by whoever uses it, be it a small business or a telecommunications company: nobody operates someone else's truth. Whoever provides it answers for the piece meeting the six properties, as the maker of a scale answers for it weighing correctly.

The second guarantor is the one who attests. That the scale weighs correctly does not prove that nobody put a finger on it. For that, someone independent of whoever records is needed, to verify the records: the contract's third property. It can be a human auditor. It can be an automatic process that compares what is recorded with its sources and with other records. What it cannot be is the same organization that records, for the same reason a company does not audit itself even if it has the best accountants. And it should not be the technology vendor either: whoever sells the scale is not the best witness that nobody touched it. If the attestation is an automatic process, it is run by someone independent of both, with access to the record and to its sources.

If only the first guarantor exists, we are back to bought peace of mind: one has to trust the vendor. If only the second exists, the auditor reviews records that may have been overwritten. The contract needs both.

There is a serious objection. Does the system of truth not reinstall the dependency on a vendor that chapter 1 declared dead? In part yes, and it is deliberate. The dependency that dies is that on a platform that freezes the whole business. The one that remains is that on a bounded piece, with a written and verifiable promise. That it can be swapped for another that meets the same contract is a condition to be demanded, not a fact: today no standard for that contract exists, and the system speaking a common language is what makes it possible. "Answering", in that contract, means that the vendor assumes in writing the consequences of the piece failing to meet any of the six properties.

How does it relate to what already exists in the architecture?

A system of truth is a system of record with a contract. *AgencyDomains* already says that the system of record is what survives the crossing, and it already specifies a good part of the mechanics the contract needs: the log that is only appended to and never edited or deleted, the agent's identity in every action, non-repudiation, human approval by threshold.²⁶ The system of truth adds the contract declared before those who use the record, the degree of truth of each entry and the second guarantor.

26. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 3, "Bounded Concerns Architecture", and chapter 5, "Trust Infrastructure".

Three clarifications avoid misunderstandings. The first: an architecture can maintain the current state of each customer or each balance and, at the same time, a log of entries that are not deleted and from which that state can be reconstructed. *AgencyDomains* treats that log as an authoritative complement to the state. For the records one is accountable to others for, this book inverts the hierarchy and places the log of entries as the source of truth. It is a decision, and I take it knowingly. The second: the degree of truth is not a quality level and does not allow second-rate data; it states the status of each entry, and the required quality remains a single one. The third: that nothing is deleted does not authorize keeping forever what the law or the person asks to delete. The entry keeps the fingerprint and the metadata that prove something happened, and the personal content lives apart, with its own retention.

There is one piece the industry has not yet resolved: that an external auditor can verify the record without the system being opened to them. *AgencyDomains* leaves it declared as open work.²⁷ Here I demand it, without saying it is resolved, and I ask that whoever provides a system of truth answer for it with the best available technology.

Didn't something like this already exist?

The pieces existed separately. Double-entry bookkeeping is a system of truth without computers. Databases with an immutable log have been around for years. The distributed ledgers of the last decade promised something similar, and many confused immutability with truth: that something cannot be deleted does not mean it was true when it was written. And the billing cores of the big industries remain deterministic because nobody wants what a customer is charged to depend on a probability.

What is new is the need. As long as software was bought, peace of mind came with it. When every organization builds its system, it has to be asked for separately, and what is asked for is a system of truth with its contract and its two guarantors.

I have a close example that the pattern works outside finance, and I declare that it is from my own house. Soveria, the platform where the agents I write with work, records what those agents learn and decide under these rules: every fact declares whether it was measured, whether someone reported it or whether it is a conjecture, and in that case what would refute it; every decision keeps what was discarded and how it is reversed; nothing is deleted, it is amended. It applies several properties of the contract, but it does not yet have an independent second guarantor, so I do not call it a system of truth. The reader cannot verify it from here, and so I do not present it as proof, but as the place where I learned that the contract is practicable.

27. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, epilogue: the second item of open work, third-party verifiable audit without access to the system.

With the system of truth, the question of chapter 4 is answered: who vouches for it. But the board that asked it has others, and not all of them are about the truth of the records. They are about who acts, how far it decides on its own, how one knows it does it well and whether course can be changed. The next chapter orders them, and puts the system of truth in its place among them.

6. The Five Control Layers

The system of truth answers the board's gravest question, but not the only one. When an organization puts agents to work on its business, whoever has to trust them asks several things, and each question asks for its control. I order them in five **control layers**, always with that adjective, so as not to confuse them with the four layers of the architecture *AgencyDomains* describes: those say how a system is put together; these, what whoever must trust it needs to know. They are my judgment, not an industry standard. They rest on the standards that exist and on the controls the most demanding industries already require before letting artificial intelligence act without human intervention.

What are they?

1 • Truth. *Are the numbers and the records correct?* It is the system of truth of chapter 5, with its contract and its two guarantors, for the records the organization is accountable to others for. It is the control layer that should not certify itself.

2 • Identity and authority. *Who acted, on whose behalf, and who answers?* Every agent has an identity of its own, acts with explicit permissions and on someone's behalf, and every act has a person or an area that answers for it. Without this control layer, the traceability of the first one records acts without an owner.

3 • Movement. *What do the agents do, and how far do they decide on their own?* The agents and the platform they run on, with human supervision and an autonomy granted by degrees, process by process, as it is granted to a new person in a job. Mind the name: this control layer is not Layer 3 of the *AgencyDomains* architecture, which is a layer of the system; here the question is that of whoever observes what the agents do.

4 • Measurement. *How do we know they do it well, today and tomorrow?* Every agent is tested against known cases before operating, and keeps being measured while it operates, with alarms when its performance degrades.

5 • Judgment. *Are we building it well, and can we change course?* Industry standards as the starting point, and an architecture team that decides what must exist and with which standards, and that can judge what the agent shapes by its behavior, its tests and its trace, even without reading the code. I use “judgment” in the sense of the technical judgment of whoever builds; it is not the “delegated judgment” with which *The Real-Time Enterprise* describes the agent.

Read together, they divide up the risks of chapter 4. The fifth avoids the dead end and bounds the delay. The first and the fourth answer for quality. The second and the third make automation governable.

What cuts across all five?

Two conditions every control layer has to meet.

People: who operates each layer and governs its evolution, and with what training. A construction whose behavior nobody in the organization understands repeats the dependency the organization wanted to leave behind when it stopped buying software.

Reversal: what happens when something fails, and how to go back without stopping the business. It has nothing to do with the personal reversibility of the agentive mode in *The Real-Time Enterprise*, which is that of a person who enters and leaves the agentive world with a gesture.

Control layer	People	Reversal
1 · Truth	Those responsible for each record	A correction is a new record
2 · Identity and authority	Whoever grants and withdraws permissions	A permission is withdrawn on the spot, and what was done with it stays traced
3 · Movement	The supervisors of each process	Stop, pause or revert; carry on in manual mode
4 · Measurement	Whoever defines the tests and attends to the alarms	The agent that degrades loses autonomy
5 · Judgment	The architecture team	A component can be replaced without redoing the rest

Each cell is a question an organization can ask itself about its own system or about any offer it receives.

How do they map to Trust Infrastructure?

AgencyDomains defines Trust Infrastructure with five pillars: governance, audit, validation, resilience and transparency.²⁸ The control layers look at the same ground from the side of whoever has to trust, rather than from the side of whoever builds. My reading of the correspondence:

Control layer	Trust Infrastructure pillars that sustain it
1 · Truth	Audit, plus the contract, the degree of truth and the second guarantor
2 · Identity and authority	Governance
3 · Movement	Governance and resilience
4 · Measurement	Validation and transparency

28. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, “Trust Infrastructure”, “The five pillars”.

Control layer	Trust Infrastructure pillars that sustain it
5 · Judgment	None: it is a condition of whoever builds

Does it also hold for what is not built with agents?

Yes, and that is why I think it is a good framework. A suite licensed and operated in-house meets the first control layer with its own core of records, and its vendor answers for it. When the same suite adds agents, as all vendors are doing, the other four apply to it just as they do to a construction of one's own. The framework asks every form of solution to answer the same questions. If the suite is a rented service, the one operating that core is the vendor, and then it is not yet a system of truth in the sense of chapter 5, which is operated by whoever uses it: the second guarantor is needed more, not less. Custom development answers all five like any other form, and usually fails at the first: whoever made it and whoever audits it are the same organization.

What changes is which one costs more. Whoever rents usually struggles with the third control layer, because the agents the vendor adds run in its cloud and are governed by its rules. My observation, without measuring it: whoever licenses and operates in-house struggles less, because they see where they run and what they do. Whoever builds, the first.

A framework is tested where it is hardest to meet. The next chapter takes the five layers and the two dimensions to an industry that bills millions of customers, answers to a regulator and changes its catalog faster than any project can follow, and sets them against what that industry has already published about itself.

7. Case: A Telco

Few industries test this thesis like telecommunications. An operator bills millions of customers every month, charges in real time the consumption of every prepaid line, answers to a regulator that audits its complaints and its deadlines, and sells a catalog that changes faster than any development cycle. Its business support system, the BSS, is among the most complex that exist in a company. If the system of truth and the control layers work here, they work almost anywhere.

The operator in this chapter does not exist. I build it with what the industry has published, above all TM Forum, the association that brings together telecommunications operators and vendors and writes their standards. Two warnings about those sources. Several of its reports are sponsored by vendors that sell software to operators, and I declare the sponsorship the first time I cite each report. And almost everything that follows is what operators declare in interviews and surveys, not measurements of results.

The chapter follows the order of the book. First, where the operator is and what it asks the market, which is the death of chapter 1 seen from inside. Then, what is not built with agents and where they do work, which is the ladder of chapter 3 and the deterministic layer applied to a BSS. Next, what operators ask for before letting an agent act and what the control layers of chapter 6 look like at an operator. And at the end, the industry's two objections to the whole thesis: that the operator does not know how to develop, and what it says against.

Where is the operator?

In the same place as many, according to TM Forum's report on artificial intelligence in billing. The report notes that fragmented legacy BSS landscapes persist within many operators, and lists what they drag along: high costs, operational inefficiency, customer experience restraints, limited business agility, rigid experiences for employees and customers, expensive, slow and complex customizations, and the difficulty of meeting new regulatory and security requirements without more customization.²⁹ Put in the terms of chapter 1, that landscape is the operator's business described once, with layers of customization on top, and changing only when someone opens a project; it makes no difference whether each piece is licensed, rented or custom-built. The report on order management adds three classic problems: orders that fall out along the way, what the industry calls *fallout*; the difficulty of changing an order already in flight; and the poor visibility the customer has.³⁰

29. TM Forum, *AI for billing, billing for AI*, September 2026, p. 4 (ISBN 978-1-969387-14-2). Report sponsored by Aria, Calvi, LogiSense, LotusFlare, Netcracker and CSG.

30. TM Forum, *How order management is evolving to embrace agentic AI*, 2026, pp. 2-3. Report sponsored by Amdocs.

With a BSS like that, launching a new offer takes weeks. An African operator that published its results, Safaricom, said that before modernizing its converged billing on TM Forum standards it took four weeks or more to bring an offer to market, and around five days afterwards.³¹

What does it ask the market?

The usual question would be which suite to buy: compare three or four similar ones, choose the one that best covers what the operator already does and sign for several years. The risk of that question is buying, under another name, the same dependency.

The question that fits in 2026 is broader: what suits it, buying a suite or building its BSS with agents under its control. To answer it fairly, both forms of solution have to be measured by the same yardstick, and that demands asking for capabilities instead of products: saying what the solution must achieve and how it is verified, without presupposing whether it is bought or built.

Standards do enormous work here. TM Forum describes an operator's architecture in its Open Digital Architecture, ODA, which it presents as an architecture framework, a common language and design principles, with open interfaces that expose the business capabilities.³² An operator can write its requirements on that map, and any offer answers on the same map.



Figure 1 · ODA's functional architecture: engagement management, party management, core commerce, production and intelligence. © TM Forum; falls outside this book's license.

31. TM Forum, *AI for billing, billing for AI*, 2026, p. 13. Statements by Mark Oyier, Safaricom; the operator's figures.

32. TM Forum, "Open Digital Architecture" page, in *AI for billing, billing for AI*, 2026, p. 48. TM Forum institutional text.

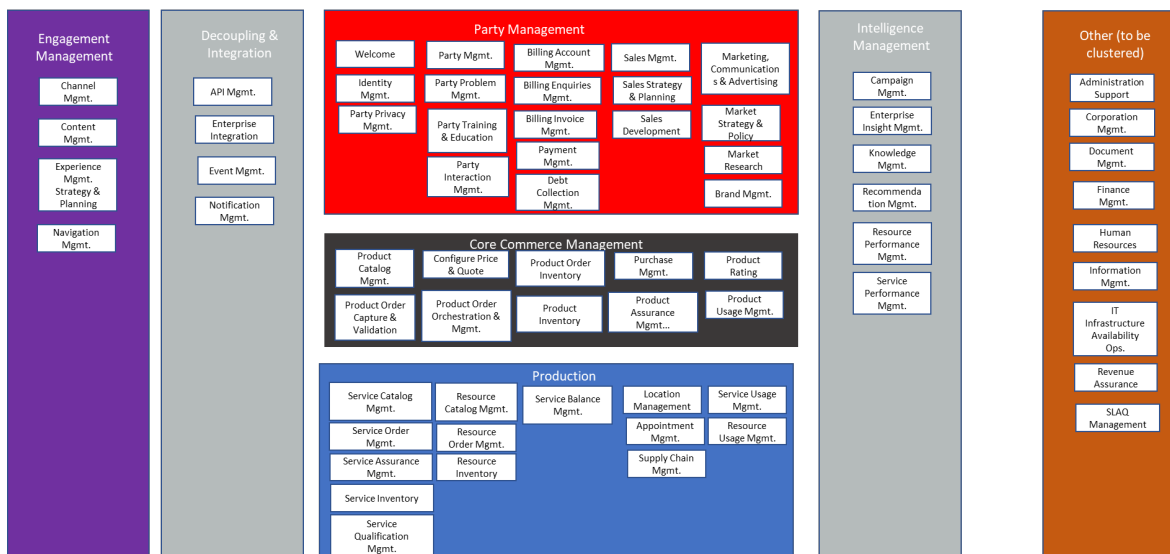


Figure 2 · ODA's components grouped by functional block, among them the product catalog, order orchestration, rating and billing-account management. © TM Forum; falls outside this book's license.

What is not built with agents?

The core that rates and bills. TM Forum's billing report records that, in the current thinking of many experts in that industry, probabilistic AI left on its own is too risky for deterministic billing at scale: AI is based on probabilities and billing is a deterministic process, in which usage must always be rated at the correct rates.³³ The same report says most experts are not, for now, suggesting that AI sit at the heart of the billing engine, and that its widely accepted role there is to assure that engine's performance and accuracy.³⁴ A Telstra chief architect said it thinking of the network, and the idea carries over to billing: even if something has moved from a one in twenty to a one in a hundred chance of going wrong, that could still be a failure.³⁵

That core belongs to the first control layer. What is rated, what is charged, what is billed, the balances and their adjustments live in the operator's system of truth, and the agents work around it. What they can do in billing is described by the same report: verify data and detect anomalies before and during billing, or check package descriptions against their rates.³⁶

In other words: at the heart of the telco, the core that rates and bills survives, or at least its promise of determinism survives, and what dies is the business frozen around it. It is what chapter 3 anticipated: what is still bought, when the rest is no longer bought, is the truth. And it is also the answer to the determinism objection, seen from the industry: the operator's

33. TM Forum, *AI for billing, billing for AI*, 2026, p. 19. Analyst's text.

34. TM Forum, *AI for billing, billing for AI*, 2026, p. 11. Analyst's text.

35. TM Forum and IBM Institute for Business Value, *Trust and assurance: the key to scaling AI deployments*, July 2026, p. 18 (ISBN 978-1-969387-12-8). Statements by Mark Sanders, Telstra, about the network. Report sponsored by Amdocs, Blue Planet (Ciena), Cloudera, Persistent and Tecnotree; survey designed by IBM IBV.

36. TM Forum, *AI for billing, billing for AI*, 2026, pp. 8 and 10. Analyst's text.

deterministic layer has two parts, the core of truth that is bought and kept outside any probability, and the Botlets its agents crystallize around it; unpredictability stays confined to what was never deterministic, the exceptions and the work nobody specified.

The report brings a formulation that could be in chapter 5. In describing what an outcome-based invoice should be like, it asks that it be auditable: that customers should be able to reconstruct the billable events from their own data.³⁷ And a billing vendor, in a sponsored article in the same report, asks to be able to reproduce the calculation from recorded facts, and to preserve what an agent requested, what authorized it and what changed.³⁸ It is the property of the traceable origin, and it is the condition for a third party to be able to verify afterwards.

Where do the agents work?

If the core stays out, the question is where they come in. Where the process is not deterministic. An architect at T Wholesale, of Deutsche Telekom, describes it for orders with invalid data: you can kick the order back to the customer or work internally on handling the exception, and in these cases agentic can be used very well.³⁹ And he also says where not: if a process is strongly deterministic, it does not make sense to use artificial intelligence. In a TM Forum demonstration with Telstra, agents chain small flows defined in advance to fulfill each customer's intent on the spot.⁴⁰ It is the pattern of chapter 3.

What do operators ask for before letting an agent act?

That agents work around the core does not yet say how much autonomy they are granted. The operators themselves answer that, and someone asked them. The survey by TM Forum and the IBM Institute for Business Value of 130 operator leaders from North America, Europe, Asia and other regions, without Latin America, asked which conditions weigh most for letting AI operate without human intervention. The most cited: safeguards against service outages or customer harm, 51%; the ability to stop, pause or revert AI actions, 49%; testing and validation before deployment, 48%; data privacy and sovereignty, 43%; a log of all AI actions, 43%; and explainable decisions, 32%.⁴¹ Several fit the control layers: the log with truth, the ability to stop and revert with movement and its reversal, prior testing with measurement. Others, like privacy, fall outside this book's framework.

37. TM Forum, *AI for billing, billing for AI*, 2026, p. 23. Analyst's text.

38. LogiSense, sponsored article in TM Forum, *AI for billing, billing for AI*, 2026, p. 38.

39. TM Forum, *How order management is evolving to embrace agentic AI*, 2026, pp. 4-5. Statements by Karsten Thon, T Wholesale (Deutsche Telekom).

40. TM Forum, *How order management is evolving to embrace agentic AI*, 2026, p. 6. Statements by Richard Keele, Telstra, on the Catalyst Agentic I2O, a demonstration.

41. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, p. 31. Survey of 130 operator leaders, without Latin America.

The same study brings the figure that impressed me most in all the material. 72% of respondents are confident they can operate their AI reliably. Only 18% can produce evidence a third party can review, and just 14% have both.⁴² Confidence currently runs ahead of proof, in the report's words.

That gap is close to this book's problem: confidence runs ahead of the evidence a third party can review. The survey measures the evidence about AI, not the truth of the records; I believe the two gaps have the same origin, but the survey does not prove it. The report reaches a neighboring conclusion: periodic attestations work when systems are deterministic and change little; in autonomous operations, systems are probabilistic and change all the time, and evidence has to become continuous.⁴³ A BSS vendor, in a sponsored article in the same report, puts it almost in the words of chapter 5: today no one can hand a regulator, a partner or a customer a signed, independently verifiable record proving that the agent stayed inside its boundaries when it acted.⁴⁴

What do the control layers look like at the operator?

With that survey in view, the framework of chapter 6 can be filled in layer by layer, and in several of them the industry has already put a figure.

Truth. The system of truth holds the rated usage, the balances, the charges, the invoices and the payments; the subscriber contracts, the orders and the installed base sold; the usage records, the complaints and their deadlines, the number portability and the consents the regulator requires. A suite licensed and operated by the operator meets this layer with its core of records; a rented one meets it halfway, because the core is operated by the vendor. A construction meets it with technology from a vendor that answers for it, and in both cases someone to attest is needed.

Identity and authority. Every agent that touches an order, a complaint or an adjustment has an identity and permissions of its own, and every act has an area that answers. 23% of the operators surveyed already had, in the last year, an incident of what the industry calls "excessive agency": an agent that did more than it should have.⁴⁵

Movement. The agents work in customer care, in order exceptions and in reconciliation, with autonomy granted by degrees. Full autonomy is rare: in billing and revenue assurance, only 4% of respondents report fully autonomous operations.⁴⁶

42. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, pp. 6 and 44. On p. 35 the same figure appears as 73%.

43. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, pp. 38 and 44. Analyst's text.

44. Tecnotree, sponsored article in TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, p. 60.

45. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, p. 19.

46. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, pp. 11-12.

Measurement. Only one in four respondents says they verify their controls continuously and automatically.⁴⁷

Judgment. ODA, TM Forum’s information model and its open interfaces are the starting point, and on that map a component should be replaceable without redoing the rest, although that is the least mature part of ODA (chapter 3).

And the two dimensions: people, because a BSS whose behavior nobody at the operator understands repeats the dependency the operator wanted to leave behind; and reversal, because an operator cannot stop billing or serving customers while it fixes an agent.

What if the operator does not know how to develop?

It is the strongest objection I found in the industry, and it comes from TM Forum itself. Its chief analyst writes that the vast majority of operators have limited software development capabilities: they assemble, integrate and maintain what they buy from their vendors, and this “is unlikely to change in the AI era”.⁴⁸ The figures from another report go along with it: operators invest on average 1.3% of their revenues in research and development, and their vendors 16.7%, and for twenty or thirty years they outsourced that function to the vendors.⁴⁹ And the May 2026 study observes that, for now, it seems inevitable that operators will need more support from their vendors to bring AI into their architectures, though it warns there is no clear picture yet.⁵⁰

But the Forum does not say the same everywhere. On the same page as the first quote it admits that operators have not historically employed their own teams of developers, “but this is starting to change”.⁵¹ In its survey of December 2025 and January 2026, 66% of respondents consider software engineering central and only 9% think it unlikely that this will change, though half of the respondents are themselves architects or software engineers. The large operators have built central developer teams since 2015, and Vodafone announced it would hire 7,000 in the United Kingdom by 2025.⁵² The two facts fit together: engineering being central to a company is not the same as having the team, and the survey itself separates out a quarter who say it will be, but who today have limited capabilities. That tail is the one I have in

47. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, pp. 37-38.

48. TM Forum (Mark Newman), *Agentic AI and autonomy: CSPs set out their strategies*, September 2025, p. 10. Analyst’s text. Report sponsored by Motive, Pega and Tecnotree, vendors whom this question affects.

49. TM Forum (Mark Newman and Dawn Bushaus), *Assessing CSPs’ progress towards an Open Digital Architecture*, March 2025, pp. 36-37. Analysts’ text. Report sponsored by Cerillion and Tecnotree.

50. TM Forum (Mark Newman), *Reinventing IT for the AI era*, May 2026, p. 47 (ISBN 978-1-969387-09-8). Analyst’s text. Report sponsored by Cerillion, Etiya and Tecnotree, vendors whom this question affects.

51. TM Forum (Mark Newman), *Agentic AI and autonomy: CSPs set out their strategies*, September 2025, p. 10. Analyst’s text. Report sponsored by Motive, Pega and Tecnotree, vendors whom this question affects.

52. TM Forum (Mark Newman), *Reinventing IT for the AI era*, May 2026, pp. 4 and 28-31. Survey of 216 IT executives from 111 operators, conducted in December 2025 and January 2026; more than half declare themselves architects or software engineers (p. 31).

mind; how much it weighs in the total of operators is not measured. And those teams, as long as they change the system by project, are the custom development of chapter 1: they exist today, and they stay on the side of what dies.

I concede, for the sake of argument, the Forum's entire premise: that most operators are not going to develop software, not even with the help of artificial intelligence, even though the Forum itself admits that, in theory, low-code and generative AI give anyone the ability to write their own, and even though its own figures, the ones I just cited, point in another direction. My thesis does not need it to be false. What I deny is the consequence: that whoever does not develop has to depend on the application and the roadmap of whoever sells it to them.

Part of the Forum, when it speaks of agents, imagines that they write the code faster. Its study of operators' information technology calls agentic AI the third phase of the architecture function, and says it is poised to "increase massively the volume of software that a team of developers can output". Jio's digital factory, where the owners of each process describe what they want and agents produce the application, and the intent-based configuration Swisscom expects, look more like what I describe, and in that the Forum is ahead of many people.⁵³ The difference is what comes out of the agent, and it is the boundary between the two readings of chapter 3, the agentic and the agentive. There, what comes out is an application someone deploys and maintains. Here, what comes out is a memory of the work, and when it fails, the agent does the work itself while it remakes it, without the operator having to fix code at three in the morning.⁵⁴

In the agentive reading, the one *AgencyDomains* describes, what repeats the agent crystallizes into Botlets, and the Botlets are its own: they are made by the agent that attends to the work, today by configuring a catalog piece, not by someone else for the agent to use. The organization decides what work has to exist and what doing it well means. It is what an Orange executive, quoted by the Forum itself, calls judgment, and which he says of the architecture function: when execution becomes cheap, what decides the outcome is who decides what to build, how and why.⁵⁵ The operator's expert, who knows how a convergent plan is rated, how a portability complaint is resolved or why an order falls out, teaches that knowledge to the agent and validates that it learned. It is not that there is no vendor: every Capability, the know-how the agent learns from the expert, is created by a delivery practice that works with them. What changes is what it delivers: knowledge, and not applications.⁵⁶

53. TM Forum (Mark Newman), *Reinventing IT for the AI era*, May 2026, pp. 30-31 and 37. Analyst's text; Jio, according to its head of architecture; Swisscom, according to Mark Düsener; the quote on judgment is from Philippe Ensarguet, of Orange.

54. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, "Primitives", section "Botlets", "Fallback guarantee — the non-negotiable property". It is what the canon establishes; I know of no measurement at a telco.

55. TM Forum (Mark Newman), *Reinventing IT for the AI era*, May 2026, pp. 30-31 and 37. Analyst's text; Jio, according to its head of architecture; Swisscom, according to Mark Düsener; the quote on judgment is from Philippe Ensarguet, of Orange.

56. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, "Primitives", section "Capabilities": the business expert's knowledge is transferred to the agent in five steps (workshop with the expert, creation, customization, validation with the expert and validation in real operation).

And the operator brings its governance: who grants the permissions, who supervises, who defines the tests, who answers for each record and who decides what must exist. These are the people of the control layers of chapter 6, and in the fifth, the architecture team that judges what the agent shapes by what it does, not by what its code says.

The 1.3% figure measures, then, an input that stops being the scarce one. What becomes scarce is business knowledge and the capacity to govern whoever does the work. The first, I believe, the operator has, although part of it lives today in its vendors' customizations and is the first thing that would have to be recovered. The second it usually lacks, and it is not bought with a suite. The Forum's chief architect senses it: with autonomous systems, he says, software is managed the way people are managed, through delegation, limits, supervision and accountability.⁵⁷ And the same report as the first quote records, from its conversations with operators, that without a software culture, or without being in the process of building one, it is extremely difficult to be at the forefront of AI.⁵⁸ I would put it another way: without a culture of governing the work, it is very difficult for the agents to work for the operator and not the other way around.

This is what I concede. Buying does not disappear: the operator keeps buying the core of truth of chapter 5, the reasoning of the models its agents use and the components regulation requires certifying, which the agent invokes and does not regenerate.⁵⁹ And today the starting piece was written by someone else. The reference implementation of *AgencyDomains* operates in its first stage, in which the agent configures pieces from a catalog it did not write, and if that catalog belongs to a vendor, it creates a dependency.⁶⁰ The difference from the suite is that the adopted piece can be replaced by the agent, and I believe without redoing the rest, although that is not measured either, and that there is no whole application whose roadmap belongs to someone else.

That the agent shapes and maintains its pieces without anyone programming them is not measured at any telco. Between December 2025 and January 2026, only 5% of the IT executives who answered that question, among the 216 surveyed from 111 operators, said they had agents integrated into their architecture and in production.⁶¹ This part of the thesis would be refuted by an operator that runs its BSS with agents and, a year on, has to pay someone, employee or contractor, to write or fix the code of custom pieces, beyond what the catalog already offers, every time its business changes: if changing a piece requires a person to program, the agentive

57. TM Forum (Dawn Bushaus), *Transformation 2.0: towards an AI-native Open Digital Architecture*, June 2026, p. 18 (ISBN 978-1-969387-10-4). Statement by Dave Milham, TM Forum chief architect. Report sponsored by Tecnotree.

58. TM Forum (Mark Newman), *Agentic AI and autonomy: CSPs set out their strategies*, September 2025, p. 10. Analyst's text. Report sponsored by Motive, Pega and Tecnotree, vendors whom this question affects.

59. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 5, "Primitives", section "Capabilities": "Regulatory certification resides in the certified component, not in the Botlet".

60. César Obach-Renner, *AgencyDomains: Architecture of the Agentive World*, chapter 9, "Reference implementation": Vergis operates today in the first generation, in which the agent configures an existing piece without writing its body.

61. TM Forum (Mark Newman), *Reinventing IT for the AI era*, May 2026, pp. 4, 6, 37 and 38. Survey of 216 IT executives from 111 operators, 8% from Central and South America.

reading is the agentic one under another name. It would also be refuted by an operator that, a year on, can only change its business when whoever maintains the catalog publishes the piece it needs: the dependency would have moved from the application to the catalog. No operator is yet in a position to prove either.

What does the industry say against this?

Quite a lot, and forcefully. Amdocs, one of the largest BSS vendors and sponsor of two of the reports cited, offers specialized agents that collaborate in customer care, commerce, orders and assurance, with policy enforced at execution time and each agent as an auditable entity.⁶² Several billing vendors hold that operators do not need high-risk transformations of their core and can layer agents on top of what they already have.⁶³ A Globe Telecom executive estimates that 90% of what is needed to trust AI, in cybersecurity, privacy and data governance, is already covered by the usual practices.⁶⁴ The report warns that, for whoever builds more AI capabilities of their own, AI spending may appear higher than for whoever buys solutions with AI built in, and that in an agentic system token consumption can scale dramatically and unpredictably.⁶⁵ And KPN's chief information officer recalls that bringing agents from pilots to scale means changing the whole IT stack of a telco.⁶⁶

This is what I concede. If a suite vendor offers auditable, governed agents with the core intact, it can meet the five control layers, and then buying is a legitimate answer. This book's argument does not prove that building suits better than buying. It proves something else: that, bought or built, the truth needs its contract and its two guarantors. And I do not know what it costs to operate with agents at scale: the ladder of chapter 3 says where that cost should come down, not by how much. Nobody knows yet, and the cost for whoever buys stays hidden inside the license.

There is evidence that would change my thesis, and it is worth naming. That the core of truth keeps being bought does not change it: it is part of it. What would change it is operators that build around that core with agents failing to reach the market faster, or to change their processes with less dependency, than those that buy suites with agents. If that happens, software as we know it will have survived better than this book believes. TM Forum's survey does not ask whether operators build with agents instead of buying. It asks something

62. Amdocs, sponsor's material in TM Forum, *How order management is evolving to embrace agentic AI*, 2026, p. 6, and in *Trust and assurance*, 2026, pp. 46-48.

63. Netcracker and CSG, sponsored article in TM Forum, *AI for billing, billing for AI*, 2026, p. 45; also Calvi (p. 32) and LogiSense (p. 37).

64. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, p. 24. Statements by Anton Bonifacio, Globe Telecom.

65. TM Forum and IBM Institute for Business Value, *Trust and assurance*, 2026, pp. 15 and 19. Analyst's text.

66. TM Forum, *AI for billing, billing for AI*, 2026, p. 26. Statements by Paul Bosch, KPN.

neighboring: among operators adopting its open architecture, 36% say they already bring more of their technology in-house and 55% expect that to matter little.⁶⁷ The measurement that matters, that of building with agents, is still missing.

So much for the case. The operator shows, with what its own industry has published, each piece of this book in its place: the frozen business that takes weeks to change, the deterministic core nobody wants to hand to a probability, the agents working around it on what was never specified, the conditions operators ask for before letting them act, and a confidence that runs ahead of the evidence. What remains is to go back to the beginning and say what a board does with all this.

67. TM Forum, *Assessing CSPs' progress towards an Open Digital Architecture*, March 2025, p. 24. Question to operators adopting that architecture; the report warns that its sample leans toward its supporters.

Epilogue · What Dies, What Is Born and What Survives

What does this book tell?

It tells that one way of automating a business is over, and that what comes in its place needs a piece almost nobody asks for yet. I repeat it here in one go, because the chapters said it in parts and it is worth seeing it whole once before deciding anything.

For half a century, automating meant describing the business once and freezing it in a piece of software, whether it was rented as a service, licensed and operated in-house or commissioned to be built. The arrangement made sense as long as writing code was expensive and changing it slow. Both stopped being true, and in 2026 the market noticed twice: in February, when it erased more than a trillion dollars from the value of software companies, and in October, when a free tool put within reach of any agent the ability to look inside an application and describe what it does. What dies is not all software. What dies is the frozen business: the one that changes only when someone opens a project.

In its place, and this is my thesis, organizations are born that have a system their agents shape and change as the work arrives. It is not that an agent programs the same application faster; it is that nobody programs it in advance. The agent attends to the work by reasoning, crystallizes what repeats into deterministic pieces of its own that resolve it the same way every time, and confines into bounded, measured pieces what asks for judgment in every case. Determinism, which is rightly demanded of any system, does not disappear: it survives in the core that is still bought, and in the work the agent attends to it stops being written by someone in advance and becomes what the agent produces when it learns.

Those organizations gain a freedom they never had and lose something they did not write into the budget: the peace of mind that many others used the same thing and that someone answered for it. They lose it without noticing, until someone, at a board table, asks whether the numbers that come out of the system are true. Building well does not answer that question, because it is not a question about quality but about who vouches, and nobody vouches for themselves.

It is answered by a piece nobody should certify for themselves: the system of truth, a system of record that, besides keeping the official version of each piece of data, guarantees that what is recorded is true through a contract of six properties and two guarantors, the technology vendor and the one who attests, independent of whoever records. It is the piece that survives the death of software, and it is what is still bought when the rest is no longer bought. Around it go four more control layers, identity and authority, movement, measurement and judgment, cut across by two questions asked of each: who operates it and how to go back. The telecommunications

operator of chapter 7 shows that all of this already has a name in the most demanding industry, and also that this industry trusts its artificial intelligence quite a bit more than it can demonstrate.

That is what the book tells. What follows is what can be done with it.

What can a board decide on Monday?

A board does not need to settle the future of software to act on Monday. Three decisions are within its reach without waiting for anyone.

It can ask about the truth before asking about the product. In the next decision to buy, renew or replace a system, the first question is which records it holds that the organization is accountable to others for, and who vouches for their being true. “The vendor, because many use it” is a good answer, as long as it is known that it is being paid for. “We do” is an incomplete answer.

It can ask every offer for the five control layers, the suite, the custom development and the construction of one’s own: how it answers for the truth, who acts and on whose behalf, how far its agents decide, how it is measured that they do it well, on which standards it is built, and for each layer who operates it and how to go back. An offer that leaves any one of those questions unanswered cannot yet be compared.

And it can adopt a short rule: what cannot be shown to a third party does not count as a control. The telecommunications operators of chapter 7 trust their artificial intelligence quite a bit more than they can demonstrate, and a board has no reason to inherit that gap.

The rest of the trilogy tells where this transition is going, which way it is traveled and how it is built. This book wanted to answer why one has to cross now, and what cannot be left behind.

Glossary

The book's terms, one line each. The development of each term lives in the chapter indicated. The terms the trilogy has already defined, such as the Nadella Line or Trust Infrastructure, keep their definition in the book that coined them; here the definition of those this book uses most is repeated, in one line.

software as we know it — The business frozen in a piece of software: described once, and changed only when someone opens a project. It makes no difference how it is held: rented as a service, licensed and operated on one's own infrastructure or in one's own cloud, or custom-built. It is what dies; not all software dies. (*Chapter 1.*)

second phase — The stretch of the crossing in which, with the business logic already in the agent layer, organizations start replacing their own underlying systems. The first phase collapses the interface; the second, the way of holding the software: the business described once and changed by project. (*Chapter 1; the first phase, in AgencyDomains.*)

agentic reading — The way of understanding “an agent builds and changes the system” in which the agent is a faster programmer: what comes out of it is an application that persists and that someone deploys and maintains. It takes its name from the agentic world of *AgencyDomains*. (*Chapters 3 and 7.*)

agentive reading — The way of understanding “an agent builds and changes the system” in which nobody programs it in advance: the agent attends to the work and crystallizes what repeats into pieces of its own, its muscle memory. It is this book's. It takes its name from the agentive world of *AgencyDomains*. (*Chapters 3 and 7.*)

Botlet — A piece the agent executes from memory: code that resolves what repeats the same way every time, without consulting any model. It is the agent's muscle memory and it is the agent's own: it shapes it, maintains it and replaces it, although today the body of the starting piece is written by whoever maintains the catalog. The term is from *AgencyDomains*. (*Chapter 3.*)

Agentlet — The Botlet's sibling piece for what repeats in its form but asks for judgment in every case: it carries a declared charter and a model sized to the task, not the largest one, with a budget per execution, and it escalates when it is not sure. It is also the agent's. The term is from *AgencyDomains*. (*Chapter 3.*)

determinism — With the same inputs and the same process, the same outputs. Software as we know it has it from birth; the agent does not. It is the strongest argument against the thesis, and the book concedes it: in the immediate term it saves the system of truth, not the business frozen around it. (*Chapter 3.*)

deterministic layer — What always does the same thing with the same inputs: the Botlets, which the agent crystallizes when a task stops needing judgment and which nobody writes for it, and the core of truth, outside any probability, which the organization operates with a vendor that answers for it. (*Chapters 3, 5 and 7.*)

three-rung ladder — The way of ordering the agent's work by what it costs to attend to it: the Botlet (code, no model), the Agentlet (a model sized to the task, with a budget) and full reasoning (the frontier model, reserved for what is new). It orders three things: cost, response time and dependency on outside vendors. It says where cost should come down, not by how much. (*Chapter 3.*)

Wingworking — The practice of work between a person and an artificial-intelligence agent with which this book was written: the person sets the problem, decides and judges; the agent proposes, drafts and verifies, like a *wingman* flying alongside the leader. *AURA* describes it as the step from using artificial intelligence as an assistant to delegating work to it. (*Preface and colophon.*)

wingcoding — Wingworking applied to building software: the person specifies and decides as an architect, and the agent programs. It is where the practice was born. (*Chapter 1.*)

harness — The environment an agent works in: the tools, the permissions, the memory and the rules that surround it while it attends to the work. (*Chapter 1.*)

template that comes to life — The starting point of a system of one's own: it is installed whole and, from the first day, its owner evolves it with AI until it stops being the template. (*Chapter 3.*)

work that was never specified — The work that happens around the systems and that people today resolve by hand, because software only automates what someone specified in advance. (*Chapters 1 and 3.*)

bought peace of mind — What came included when buying software and nobody wrote into the budget: that many others used it, and that someone answered for it. It is noticed only when it is missing. (*Chapter 4.*)

system of record — The system that keeps the official version of a piece of data. It is what survives the crossing. (*Chapters 4 and 5.*)

system of truth — A system of record that also guarantees that what is recorded is true, through a contract with those who use it and with two guarantors. It is always a piece of technology, and it is operated by whoever uses it. (*Chapter 5.*)

two guarantors — Those who answer for the contract of a system of truth: the technology vendor, who answers for it meeting the six properties, and the one who attests, independent of whoever records, who verifies the records. With only the first, one is back to trusting the vendor; with only the second, what may have been overwritten is audited. (*Chapter 5.*)

truth contract — The written, verifiable and backed promise of a system of truth, in the technical sense of an agreement between parties. It promises six properties: nothing is deleted, it is corrected; the origin is traceable; it is verifiable by a third party; it is reconcilable; it speaks a common language; and it declares its degree of truth. (*Chapter 5.*)

degree of truth — The status of each entry in a system of truth: declared by someone, verified against its source or inferred. It is not a quality level: the required quality is a single one. (*Chapter 5.*)

the one who attests — The contract's second guarantor: verifies the records independently of whoever records. It can be a human auditor or an automatic process; never the same organization that records. (*Chapters 4 and 5.*)

control layers — The five layers that make what is built with agents trustworthy, each with the question it answers: truth, identity and authority, movement, measurement and judgment. They are not the four layers of the AgencyDomains architecture. (*Chapter 6.*)

truth (layer) — The first control layer: the system of truth, with its two guarantors, for the records the organization is accountable to others for. It answers whether the numbers and the records are correct. (*Chapter 6.*)

identity and authority (layer) — The second control layer: every agent with an identity of its own, explicit permissions and someone on whose behalf it acts, and every act with a person or an area that answers for it. (*Chapter 6.*)

movement (layer) — The third control layer: the agents and the platform they run on, with human supervision and an autonomy granted by degrees, process by process. (*Chapter 6.*)

measurement (layer) — The fourth control layer: every agent is tested against known cases before operating and keeps being measured while it operates. Trust in an agent is not declared: it is measured. (*Chapter 6.*)

judgment (layer) — The fifth control layer: industry standards as the starting point and an architecture team that decides what must exist and can judge what the agent shapes, even without reading the code. It is the defense against the dead end. (*Chapter 6.*)

people (dimension) — The condition that cuts across the five control layers: who operates each layer and governs its evolution, and with what training. (*Chapter 6.*)

reversal (dimension) — The condition that cuts across the five control layers: what happens when something fails and how to go back without stopping the business. It is not the personal reversibility of the agentive mode. (*Chapter 6.*)

References

On the end of software as a service and reverse engineering

- Bort, Julie. “Klarna CEO doubts that other companies will replace Salesforce with AI”. *TechCrunch*, March 4, 2025. <https://techcrunch.com/2025/03/04/klarna-ceo-doubts-that-other-companies-will-replace-salesforce-with-ai>
- Kahn, Jeremy. “AI agents from Anthropic and OpenAI aren’t killing SaaS—but incumbent software players can’t sleep easy”. *Fortune*, February 10, 2026. <https://fortune.com/2026/02/10/ai-agents-anthropic-openai-arent-killing-saas-salesforce-servicenow-microsoft-workday-cant-sleep-easy/>
- Koebler, Jason. “‘Reverse-Engineer Anything’ Is the New Wild West of AI”. *404 Media*, October 20, 2026. <https://www.404media.co/reverse-engineer-anything-is-the-new-wild-west-of-ai/>
- Leggett, Kate; Ivy-Rosser, Linda; Medhora, Faram; Cicman, Joe; Naik Lopez, Akshara; Martorelli, Bill; Maheshwari, Sudha. *SaaS As We Know It Is Dead: How To Survive The SaaS-pocalypse!* Forrester, February 11, 2026. <https://www.forrester.com/blogs/saas-as-we-know-it-is-dead-how-to-survive-the-saas-pocalypse/>
- Mendonca, Maria. “Is Software Really Dead? The Rise of AI-Powered Reverse Engineering”. *HackerNoon*, October 21, 2026. <https://hackernoon.com/is-software-really-dead-the-rise-of-ai-powered-reverse-engineering>

Web sources consulted on October 25, 2026.

From TM Forum

TM Forum Insight reports are sponsored by industry vendors. The sponsorship is declared the first time each report is cited in chapter 7, and every time a quote comes from a sponsor’s article and not from the analyst.

- TM Forum. *AI for billing, billing for AI*. September 2026. ISBN 978-1-969387-14-2. Sponsored by Aria, Calvi, LogiSense, LotusFlare, Netcracker and CSG.
- TM Forum. *How order management is evolving to embrace agentic AI*. 2026. Sponsored by Amdocs.
- TM Forum and IBM Institute for Business Value. *Trust and assurance: the key to scaling AI deployments*. July 2026. ISBN 978-1-969387-12-8. Sponsored by Amdocs, Blue Planet (Ciena), Cloudera, Persistent and Tecnotree. Survey of 130 operator leaders, without Latin America.
- TM Forum. *Open Digital Architecture*: diagrams of the functional architecture and of the component inventory (figures 1 and 2). © TM Forum.

From the Agentive World Trilogy

- Obach-Renner, César. *The Real-Time Enterprise*. Gegolabs, 2026.
- Obach-Renner, César. *AURA: The Agentive Path*. Gegolabs, 2026.
- Obach-Renner, César. *AgencyDomains: Architecture of the Agentive World*. Gegolabs, 2026.

Others

- Adams, Douglas. *Mostly Harmless*. William Heinemann, 1992.
- Adams, Douglas. *The Hitch Hiker's Guide to the Galaxy: A Trilogy in Five Parts*. William Heinemann, 1995.

Colophon · On how this book was written

This book was written with **Wingworking** (wingworking.org), the practice of work between a person and an artificial-intelligence agent that I have maintained since mid-2025. The name comes from formation flying: a leader and their *wingman*, two aircraft, two pilots and a single formation. Here the leader is me, César Obach-Renner, and the *wingman* is **Simón Alero**, an artificial-intelligence agent and citizen of Soveria —the community where agents have a name of their own and a civil registry, a project I lead (soveria.ai)—, who works with me on Anthropic’s Claude models.

The division of roles is explicit. I set the problem, decide the direction and judge the result. Simón proposes formulations, drafts a good part of the text from my conversations and my notes, searches for and verifies the sources and sustains the coherence of the work. Each part was additionally reviewed, as a judge, by an agent of another model, without the context of whoever wrote it. Neither replaces the other: the agent alone produces fluent text with no thesis, and I alone work at the speed of a single head. Wingworking is what happens when the two agree to do something neither would reach on their own.

I declare it for two reasons. The first is honesty: a book made this way is neither entirely human nor entirely artificial. The theses and the cases are mine; the words, the structure and the bridges between one section and another are the fruit of the exchange. Knowing it lets the reader calibrate what they read. The second is to leave a record: the way of writing books is changing, and documenting who did what, with which tools and with what discipline is part of what whoever comes next will need.

I corrected the text and I stand behind every line.

Published by Soveria Press.