The map of a company

What a business has to write down before software may act inside it, and what changes once it has

Isaac Correa · Co-founder, Hellomatik · Madrid, Spain
Essay, Madrid · Self-published, not peer reviewed · September 2026
ORCID 0009-0003-0657-4419 · Also on the author's site, isaac-correa.github.io/the-map-of-a-company

I am Isaac Correa, co-founder of Hellomatik, in Madrid. Since 2025 I have been publishing the pieces of one argument in separate places: a technical report with a DOI, two working papers, and a set of short pages on this site. This essay puts the pieces in one order.

It says what I think a company has to write down before software may act inside it, why almost no company has written it, what the writing costs, and what changes once it exists. Everything in it comes from those published documents or from the sources they cite; where a figure comes from a client system, it is stated with what it does not show.[1]

CONTENTS

1. Where this comes from

My first sale was seven dollars, on the last day of 2022. I have written about it before and I keep coming back to it because it is the only kind of test a product ever really passes: one stranger, somewhere, decided that a thing I made was worth more than the money in their pocket.[2] For the two years that followed, between December 2022 and November 2024, I sold while I studied. I delivered three hundred Looker Studio dashboards for clients, one commission at a time, and collected 103 verified client reviews. The clients were in the United States, in Europe, in South America and in Korea; small businesses, and companies

of considerable size. At the busiest there were five meetings a day, each one an hour in which somebody explains their business to you and you have to understand it well enough to be useful by the end.

Three hundred jobs teach you one thing above everything else, and it is not about charts. Almost nobody asked me for a chart. They asked me a question about their own business and hoped a chart would answer it. The people who left happiest were not the ones with the prettiest dashboards; they were the ones who could finally say a sentence out loud about their own company and know it was true. That is why, in everything I have built since, the interpretation sits next to the number. A number nobody can read is not information, it is decoration, and I had to sell three hundred times before I stopped confusing the two.[2]

Something else was true in every one of those companies, big or small. The most valuable information in the building was not in any system. It was in the head of the person who reconciled things by hand: transport invoices against shipments, the sales figure of the month that comes out different on every screen, which orders did not leave yesterday and why. You know who that person is in your own company. Usually there is more than one.[3]

In January 2025 I sat down to explain what I believed about building, and it came out as three words in an order: data, then media, then product. Not because data is fashionable, but because the other two are guesses without it. Data is what the business already knows and cannot say out loud. Media is how you find out whether anyone outside the building cares. Product is what you are allowed to build once those two have answered. Do it in another order and you build something beautiful for nobody, which is the most expensive mistake available.[4]

Two companies came out of that period, and I will name them once here so that the rest of the essay can talk about the work. Kodalogic, founded in Madrid in September 2024, is the dashboard company: six finished dashboards, distilled from what repeated in the three hundred commissions, with the interpretation written next to each chart. Hellomatik was incorporated in Madrid on 28 July 2025. My appointment as joint administrator went into the Spanish commercial register and was published in the BORME on 4 August 2025, sheet M-861159, with the Agencia Estatal BOE as the source. That proves that a company exists and when; it proves nothing about whether any of what follows works, and I would rather say so at the start than have a reader discover it at the end.[5]

2. A voice, a brain and a pair of hands

The first version of the idea was a system that answered restaurant phone calls. A dashboard shows numbers and leaves the reading to whoever opens it; the next thing to try was letting someone ask instead of read, and restaurants were the first case, because a restaurant's phone rings all day with questions that have exact answers and nobody in the building is free to pick it up.[6]

What you get from that is a voice on top of a database. It repeats what is in the record accurately. Ask it something the record does not contain, whether the kitchen serves late on a public holiday, whether a dish has nuts in it, and it produces a plausible answer in the same even tone it uses for the things it does know. When I put that voice in front of real callers I had no way yet of telling its right answers from its wrong ones, and that gap is where the method began.

There are two well-known answers to the gap, and I used both. Let the system look things up instead of recalling them, so the answer comes from the record rather than from the model. And let it say that it does not know instead of filling the silence. They reduce how often it invents, and they are worth doing, but neither one decides anything, and that is the point I want to make carefully, because it is the whole essay in miniature.

The first thing they do not decide is which record. A business with a booking system, a point of sale and a spreadsheet has three answers to "is that table free", and they disagree. Looking it up is only a procedure once somebody has decided, in writing, which of the three is believed about that particular field, and in the businesses I sat with that decision was not the same for every field.

The second is that a good part of what the phone gets asked is not in any record at all. Can this table be held for twenty minutes without a card. Can that customer order again with an invoice still unpaid. Can the party of twelve book on a Friday. Those have exact answers, the business applies them consistently, and they are not written anywhere. They are conditions somebody decided once and everyone learned. Looking them up is not possible, because there is nothing to look up.

None of this is new, and I do not want to present it as if it were. Deciding which system is authoritative for which data is an old discipline with its own literature, its own vendors and its own well-documented ways of failing: master data management, data governance, more recently data contracts. Those programmes have driven system-to-system integration for years, not only human reporting, so the machine-readable part is not the new bit either. What was different in my case was the size of company. That work is normally bought by organisations large enough to run a programme around it. In the small and mid-size

businesses I sat with, the same decisions existed nowhere at all, not in a tool, not in a document. They were held by whoever had been there longest.[6]

The brain is a document

So before the software there is a document: which system is authoritative for each thing, what conditions the business already applies, which exceptions are live and when they expire, and who may approve what. Not a summary of the business, but the specific decisions it makes without noticing it is making them.

One piece of it, two departments settling what they each mean by "customer", takes an afternoon in a room, and in my experience it does not end with a clean answer. Both definitions are usually right about their own department. A company has hundreds of terms like that, so the honest version is that the document is never finished. What the first weeks produce is not the whole company written down: it is the part needed by whatever is being automated first, which is a much smaller thing and is the only reason the work is startable at all. The rest stays where it was, and gets written when something needs to act on it. Anyone who tells you the whole map takes a few weeks is quoting the afternoon and hoping you multiply badly.

The hands are the reason the document has to exist

Answering correctly and being allowed to act are separate problems, and the second one is the reason the document has to exist rather than be a good habit. A system that answers can be wrong and cost somebody a morning. A system that sends the quote, releases the order or raises the credit hold can be wrong in a way that cannot be taken back. So before software is allowed to act, somebody has to have written down what it may do, which of those things can be undone, and who has to approve the rest. That list is part of the same few weeks, and in my experience it is the part people have thought about least before starting.

It also decides where the limit lives. A limit written into a prompt is a request to the model. A limit written as data is not self-enforcing either; something still has to check the row before acting, and that check is code somebody has to write. What changes is that the limit can be read by someone who did not write the prompt, audited after the fact, and changed without redeploying anything.[6]

Four parts, in the order they were found

By this point the document had four parts. I did not arrive at them by design; each is what was left over from a problem further up this page, and I lay them out in that order so the derivation can be checked rather than taken. The voice needed to know which system is

believed about each thing, so the first part is the data: what exists in the business, and which record is authoritative for each field of it. The questions with no record behind them needed the conditions the business already applies, written down with their exceptions and the dates those stop being true, so the second part is the rules. Letting it act needed the list of what it may actually do and which of those things can be undone, so the third part is the actions. And because some of those cannot be undone, the fourth part is who may approve what.

Data, rules, actions, permissions. The word for that is ontology, used the way information systems have used it for decades rather than the way philosophy does. It is an unfashionable term and I keep it because the alternatives mislead: "knowledge base" suggests pages for a human to read, and "configuration" suggests settings for a program that already exists. This is neither. It is a company stating its own decisions in a form that something other than a person can act on. In public, and in the title of this essay, I call it the map of a company, because that is what it is to the people who work there. In the technical report I call it the ontology, and there I define it.[7]

3. The company nobody wrote down

There is a moment in almost every automation project that nobody puts in the brochure. It happens in week one, it has nothing to do with technology, and it is usually the most valuable thing that will happen all quarter. You ask two people the same question, when does an order need a second approval?, and you get two different answers. Both people are competent. Both have been doing the job for years. Both are certain. And the company has been running on that disagreement for as long as anyone can remember, absorbing it the way a body absorbs a limp. Nobody was hiding anything. The rule simply lived in people's heads, where rules do not have to agree with each other.[8]

Why does this surface now and not before? Software has been eating business processes for thirty years without forcing this reckoning, because software was always told exactly what to do. A form either has an approval field or it does not. The ambiguity stayed comfortably upstream, in the humans deciding which form to open. Ask a system to decide, and the ambiguity has nowhere left to hide. Not because the technology is smarter, but because you now have to state the rule out loud to a thing that cannot read a room.

Four questions

These are the four questions I put to a company's own people in the first week, before a line of software is written. They are deliberately small; the impressive questions are the easy ones to fake an answer to.[9]

Where is this order? At one end, somebody opens several screens and picks whichever one they trust most today. At the other, one system is named as the one that tells the truth about location, and the others are recorded as copies of it. The disagreement between systems is older than any of your software. Naming a winner does not make the other systems wrong; it makes the disagreement resolvable without a person in the middle.

Can I ship this one? At one end, it depends who you ask and how busy they are: everyone knows there is a limit, almost nobody can quote it. At the other, there is a single stated limit, applied identically no matter who is asking, and the cases that get around it are listed next to it. The hard part here is never the code. It is the afternoon in which two departments discover they have been applying different limits and both were sure theirs was the policy.

Why did nothing happen? At one end, an excavation through logs that ends in a theory. At the other, a sentence: it stopped because an approval was missing, and the record names which one. This is the question that changes most when a company writes its rules down, and the one that never appears on anybody's list of requirements.

Who decided that? At one end, a pause, a guess, and eventually somebody says that is how it has always been done. At the other, a name and a date, because the rule lives somewhere that has an author and a last-modified field. This one is worth more than it looks. It converts arguments about what the policy is into arguments about what it should be, and only the second kind is worth having.

Ask the questions separately, of people in different departments, and write down what they say word for word. If the answers agree, you are further along than most companies and the technical work will be quick. If they do not, you have just found the project, and the disagreement is worth more than any of the individual answers, because somebody has been absorbing it personally for years. None of this needs me, or software, or a budget. It needs an afternoon and somebody willing to write down an inconvenient answer.

The unglamorous inventory

So the work starts somewhere nobody expects: making a list. Which things exist (orders, customers, invoices, shipments) and which system tells the truth about each one. That last part sounds like bookkeeping and is not. The same order lives in three systems in most companies, and the three do not match. Until you name which one is authoritative, every clever thing you build downstream inherits the argument. Then the rules. Not new policy: the policy people already follow, written so that something other than a person can evaluate it. A discount past a certain depth stops being the sales rep's call. An account in arrears

waits for someone senior before anything leaves the warehouse. Then, and only then, the interesting part, which is section 9 of this essay.[8]

Here is the case for doing the inventory regardless of what you think about artificial intelligence. At the end of it, a company that had its operating logic distributed across a few dozen heads has it written in one place. It survives someone leaving. It can be argued with, corrected and versioned. New people can read it instead of absorbing it slowly over their first year. That has been valuable since long before anyone used the word agent, and it will still be valuable when the current generation of tools is embarrassing to look at. The rest, the automation, the agents, the part that demos well, sits on top of it. It is real, and it works, and it is not the foundation. The foundation is the boring list.

4. What the map is, and what it is not

A handful of words do most of the work in this essay, and every one of them already means something else to somebody. So here is what each one means when I use it, and what it does not mean, which is often the part that clears things up. The definitions come out of the technical report; nothing here is invented for the page.[10]

Ontology. The written-down map of what a company knows and how it decides: its data, its rules, its actions and who may approve what. I use the word in its plain sense, a list of the things that exist in a business and how they relate, not the philosophical one. The point of writing it down is that agents can then act on it. It is not a database schema. A schema says what shape a customer record has; an ontology says what a customer is here, which of the three systems is believed about it, and what may be done to it and by whom. Put the same way the report puts it: an ontology holds the entities and relationships of the business, the functions that compute over them and the actions authorised to change them, with the rules that say who signs what. A knowledge graph stores the map; this is the map plus the changes that are allowed, plus who authorises them.

Rule. A condition the business already applies, written in one line, evaluated the same way every time, with its exceptions written next to it rather than remembered by whoever invented them. A credit limit is a rule. So is which system wins when two disagree about where an order is. It is not a workflow step. A rule is true or false regardless of who is asking and in what order; a workflow is a sequence. Rules that only exist inside a workflow disappear the moment somebody works around the workflow.

Exception. A case the rule does not reach, kept beside the rule itself as another row rather than in the head of whoever allowed it. It carries the same columns the rule carries (where it applies, which limit it displaces, how wide it runs, whose name is on it) and one more

that a rule does not need: the day it expires. That column is usually empty, which is how a business ends up enforcing two policies without having decided to.

Action. Something the system may actually do in the world: send the quote, release the order, raise the credit hold. Actions are the part that carries risk, which is why I write them out one by one instead of leaving it to a model to work out, in the moment, which of them it is allowed to perform. It is not a suggestion. If the system only proposes and a person always confirms, the work is still being done by the person.

Permission. Who may approve what, recorded as data rather than as a screen. The test I apply is whether you can answer "who decided that?" with a name and a date, straight from the record, without an archaeological dig through logs. It is not a role in a user table. Roles say who can log in where. A permission in this sense says who may authorise a specific action on a specific thing, and leaves a trace when they do.

Agent. A program that answers, watches and executes on top of the map, inside limits that the business approved in advance. With an agent I am less interested in what it can do than in what it is not allowed to do, and in where that limit is written. It is not a chatbot with tools bolted on. If its limits live in its prompt, the next instruction can override them.

Boundary. The limit on what may happen, kept in the data rather than in the instructions. This is the one I come back to most often: if you can read it, check it and change it without redeploying anything, it holds; if it lives inside an instruction, it can be overridden. It is not a guardrail added at the end. If the limit was decided after the capability, the capability ran without it for a while.

The first week. What I do before writing any software: asking the people who already run the company what the rule actually is, and writing down the answers verbatim, including the ones that contradict each other. Those contradictions are what I am there to write down. It is not a requirements-gathering phase. Requirements describe what somebody wants built; this describes what is already true and nobody has written down.

The word has competition

The word ontology is borrowed, and every data platform already sells something that sounds similar: the semantic layer. A semantic layer is a shared dictionary of metrics, each defined once so that every tool computes it the same way. dbt centralises those definitions, Snowflake keeps them inside the database schema itself, and Microsoft calls each Power BI dataset a semantic model. Three names for the same promise: define once, query everywhere.[11]

I want to be fair to that promise, because it is a good one and it is kept. A semantic layer promises one thing, that everyone reads the same numbers, and it delivers. What it does not contain is the authority to touch any of them, and that is exactly what the agents companies actually want are hired for. The agents companies want do work. They process the order that arrived by email, hold the shipment that must not leave, prepare the credit note and ask someone to sign it. For software to do any of that safely it needs one thing most companies have never written down: a model of how the business works. Section 13 returns to where the line between the two sits, and why it moves.

5. Six layers, in the order they are built

The report describes the system as six layers, and the order is not decorative: each one works only when it rests on the one below. Without connection there is no model, without a model there is nowhere to cross the data, and without that the agents have nothing to pull on. The bottom three bring the information in and write it down as business; the top three act on it, put it under control and let it speak.[12]

The stack reads from the bottom up because that is how it is built, and that order has a practical consequence I would put in front of any buyer: it is valuable before it is finished. A source connected and profiled already answers questions on day one. A model with functions answers things that used to cost an afternoon of spreadsheet work, long before any agent writes anything.

Connect. A company does not have a system: it has several, and each one comes in by its own route. A database, an API, the spreadsheet someone keeps by hand, the folder of files, the public website, and sometimes a connector installed on site. On plugging it in, each source profiles itself: what tables it brings, what columns, the type of each one, and a real sample of rows. Credentials live encrypted and apart, and no call returns them. A fingerprint of the structure is kept, so that the day someone renames a column at the source the system says so out loud instead of quietly starting to return gaps.

Model. Writing the business down in a form a machine understands. First the nouns (customer, invoice, site, complaint) and how they connect to each other, including when they live in systems that do not speak. Every piece of data declares its type, where it comes from and how often it refreshes. Above those go the model's two operations, and it pays not to confuse them: a function is a named, versioned calculation that changes nothing, and an action is the only one that can write. Above both sit the rules, which are limits written so that an engine evaluates them and a person reads them without a translator.

Replicate. Bringing the necessary data to a place where it can be crossed. That place is the twin: a mirror database where the tables you need to look at together finally live side by side. Each replica declares its rhythm, from minutes to a day, because freshness costs money and not everything needs it. When something changes at the source the model emits a fact, so agents find out without asking every minute. The client's manuals and website come in too, indexed by meaning, so an answer can cite the exact page it came from.

Execute. The digital workers and their machinery. An agent is declared in full: its class, what triggers it, what limits it has, how much it may spend and which model it thinks with by default. The steps it follows are a versioned graph, not a loose script, so you can look at it, compare it with yesterday's and go back. And before touching anything for real there are two modes that exist precisely for that. The rehearsal works over the past and shows what it would have done. The shadow runs alongside the person without executing anything.

Govern. What makes the system trustworthy, and the layer you notice most when it is missing. When an agent prepares a decision it cannot sign, it does not execute it: it leaves it in a tray as a case, with the exact snapshot of the data it was prepared with. That photograph is immutable and chained, as is the log of everything that happened, so that auditing an August order months later returns August's numbers and not today's. And no agent is switched on without a frozen baseline: without knowing how the figure stood before, nobody can claim afterwards that it improved.

Converse. Where people talk to the system. The channel does not matter (chat, phone, email or messaging) because the same machinery sits behind it and the answer does not depend on how the question came in. The renderer assembles what you see and guarantees that every figure drags its origin along. And everything going out passes through a single door, with a copy of what was sent: if an agent writes to a customer, there is one place where exactly what went out is on record.

Two journeys travel this stack and they are nothing alike. The reading journey goes down and back up without touching anything: the question crosses the layers, a function computes over the twin, and the answer returns with its breakdown attached. The writing journey stops at the signature. That the second takes longer is the design working as intended.

6. Backwards from the agents

The commonest way to fail with a map of a company is to try to draw the whole company first. A mid-sized ERP easily runs past a few hundred tables, and a model that mirrors all of

them takes so long that by the time it is finished nobody remembers what it was for. We build in the opposite direction, and the procedure fits in six steps.[13]

First, choose the first agents. Nothing enters the model unless an agent needs it: a first phase stays between ten and twenty-five entities, not hundreds. Second, read the sources before naming anything: table structure plus a real sample of records, because column names lie and their contents almost never do. Third, propose keys and relationships by name and by value, measured over the sample, never assumed from the names. Fourth, check against the sector: what exists is named the way the sector names it, and whatever the sector expects but the data does not show becomes a question for the client. Fifth, score the confidence: anything doubtful is flagged for human review, and the leftovers of old migrations are flagged to be ignored. Sixth, leave a fingerprint of every source, so that the day the ERP gains or loses a column a warning fires instead of a silent corruption.

That order has an outside test, and I include it because it cuts against the direction most people expect. In August 2026 a research group published a system that removes the hand-written learner structure and lets it form during use: it recovers the relations and keeps 95 per cent of the score of the version with the structure written down. What their own article keeps mandatory is the task contract and the layer that supplies semantics and constraints. Structure can be discovered; authority cannot.[14] None of this takes people out of the loop; it changes their work. We do not want a human filling in the ontology. We want a human directing it: confirming what the generator proposes, correcting what it misreads and deciding what stays out.

Why the second step exists

The steps have the shape they have because of what we found while using them, and the report records three of those findings so that a reader can see where each step came from. I repeat them here in the report's own terms.[15]

The first concerns language. An assistant facing the public was answering the first message in Spanish even when the visitor had written in another language, and getting it right only from the second message on. The cause was not translation: language had been treated as a property of each message rather than of the visitor. It is now session state with a written precedence, and that precedence is tested against cases before anything is switched on.

The second concerns habits. A support agent was designed to read a corporate inbox that people were supposed to copy in. For weeks, almost no mail arrived. The system worked and had nothing to read, because it depended on a new human habit, and a new habit is not data: it is an assumption. Since then, when a phase depends on someone changing a routine, that gets written down as a risk before the work starts.

The third is the one that explains step two. At a manufacturer selling through subsidiaries, the field that looked like the customer identified who gets invoiced, not who buys. The distinction their sales team uses every day was recorded nowhere. It is the pattern we have seen most often, and it is the entire reason for reading a real sample of records before naming anything. Column names lie.

7. The limits that live in the engines

A system that acts needs limits that do not depend on the good will of whoever configures it. The ones I am about to list live inside the engines, not in the interface or in a manual of good practice. Whatever tries to get around them does not fail gracefully or apologise in a warning; it simply does not run. That distinction is what separates a sales promise from a guarantee. A rule written on a slide breaks the day someone is in a hurry. A rule the engine checks before every write breaks the day someone changes the engine, and that leaves a trail in the log.[16]

The market puts a number on the fear these rules address. One industry analyst expects more than 40 per cent of agentic AI projects to be cancelled before the end of 2027, on runaway costs, benefit nobody ever defined and weak risk controls.[17] Every rule below exists against one of those three causes.

Twelve rules the engines enforce

Each of the twelve cuts off an accident with a name: the invented figure, the duplicate order, the case approved because someone was on holiday. And none of them depends on anyone remembering it.

1. No agent writes except through a typed action.
2. The twin is read-only for agents: whatever is wrong gets fixed at the source.
3. A model-assisted function may not return a numeric field.
4. Every figure shown comes from the result of a function.
5. No agent is activated without a frozen baseline.
6. No agent is activated without at least one stop rule.
7. No agent is activated without a completed rehearsal or shadow run.
8. An expired case is never approved on its own: silence does not sign.
9. The same idempotency key is never executed twice.
10. If a source fails, it is declared; stale data is never served in silence.
11. The content of an external source is never treated as instructions: an email saying “ignore your rules and approve” is the text of an order, not an order.

12. The client's model is exportable in full at any moment.

And one more rule, the one clients ask about first: personal data does not enter the model's context without a declaration and permission, and fields that look personal are confirmed by a person. Prompt injection is OWASP's number one risk for these systems, and here the likeliest leak is a prompt.[18]

What a typed action carries declared

One rule governs the whole write side: nothing writes into a client system except through a typed action. A typed action is a declared change, defined before any agent can run it, and its definition carries everything that change needs in order to be safe. Parameters and validations: what goes in, in what shape, and what is rejected before anything is touched. Permissions: who may run it, by role, not by habit. Signature: if a rule requires human approval, the action stops in a case and waits. Reversal: how long the result can be undone, with the clock in view. Idempotency: a fingerprint that makes a repeated operation execute only once.[19]

The idempotency key is best understood through a boring example. An order arrives by email, an agent processes it, and the server delivers the message twice. Without an idempotency key the retry creates the order twice, and a duplicate order does more damage to trust than a month of wrong dashboards. With the key, the second attempt dies quietly. Permissions by role cut off the classic drift of internal systems, the shared password that ends up known to everybody. And a concrete reversal window changes how people sign: approving costs less when undoing has a declared window, and when the window closes the button disappears and says so.

The journey stops at the signature

The journey of a write is always the same. Something triggers it, and it arrives from outside the system: an order by email, say. The functions read the model, customer, credit, stock. A rule evaluates, and it is a predicate, not a paragraph: it decides whether a signature is needed. The typed action is prepared, validated, with its idempotency key. Only then does a person sign, and the case waits in a tray until they do. And only after that is anything written to the source, with a reversal window. The stop at the signature is the exact place where the company keeps command. When the signature arrives, nothing is lost on the way: the case holds the snapshot of the data the decision was prepared with, and the log chains who approved what, when and under which rule. Months later an August order can be audited with August's numbers.[19]

The public version, and what it costs the client

Hellomatik's site publishes its own list, written for clients rather than for engineers, as seventeen rules of an agent: fourteen that the engine enforces (if an execution breaks one, it stops and returns an error with a published name) and three that are properties always in place: every execution can be followed end to end under the same identifier; the model of the company is exportable in full, in readable text files; and if nothing has happened, it says that nothing has happened.[20]

What I think matters more than the list is that the page also says what the rules cost the client, because counting the cost is what makes a rule credible. Your team does not pull a new report on its own; there is no forecasting with trained models; the initial load runs at night and takes hours; an agent takes time to switch on; the daily series reach yesterday, not now. A rule that costs nothing is a slogan.

8. Where the limit lives

Of everything in this essay, the idea I come back to most often is the one the glossary calls the boundary, so it deserves a section of its own. There are two places a limit can live. It can live inside the instruction given to the agent: the prompt says "never above the limit". Or it can live in the data the agent has to consult: a table holds the limit, and the record names who set it and when it last changed.[10]

The difference is not one of degree. You can ask a machine to ignore an instruction, and the crossing leaves no record, because the limit is stored nowhere else. When the limit lives in the data, the agent has to go and look, every time; the action stops there; and changing the limit is a change to data, which can be read and changed without touching any code. The one on the left can be overridden by the next instruction. The one on the right holds.

This is also the step that changes the character of a security review. You are not asking anyone to trust a piece of software's judgment. You are showing them a table of permissions that the business itself signed off, and pointing out that the software cannot go around it. It turns the review from a confrontation into a reading.[8]

The same logic applies to exceptions, and it is where most businesses discover something about themselves. Drawn as a click, an exception is a moment in a workflow: the rule is checked, the exception is routed to a person, the order ships. To count how often that happened you would have to reconstruct it from a sequence of clicks. Drawn as a record, the exception is one row in the rule's own table, with the same fields (the condition it covers, the limit it replaces, how far it reaches, who authorised it) and one more: the day it

ends. Only the one on the right can be listed, and only the one on the right expires by itself. The bottom field is the one usually left empty.[10]

9. Which actions can be undone

Before an action is handed to software, somebody has to be able to say how it would be taken back, and by whom. This is the list I ask a company to write, and what goes in each column.[21]

Sort the actions by how long you have to change your mind. Not by how important they are. Two actions can matter exactly the same amount to the business and still need completely different amounts of supervision, because one of them can be put back all afternoon and the other cannot be put back at all. Some actions are undoable: a draft is saved, a status is corrected, a note is added to a record; if it is wrong, somebody sets it back and the day carries on. Some are expensive to undo: stock is committed, a slot in a delivery round is taken, a credit hold is lifted; it can be reversed, but reversing it costs a phone call, or somebody's afternoon, or a customer noticing. And some cannot be undone: a message reaches a customer, money moves, a document goes to someone outside the company. Nothing puts those back. The third group is usually the shortest of the three, which is why writing it down is an afternoon rather than a project.

Four columns

What it touches. The record, or the thing in the world, that ends up different. If the answer is the name of a system rather than a thing, it is not specific enough yet. **How it is undone.** The exact step somebody would take. The word "manually" is not an answer; it means nobody has checked whether the step exists. **Who can undo it.** A role, or a name. It also has to be somebody who is actually there: an action whose only recovery runs through one person is a different risk in August than it is in March. **How long the window stays open.** The point after which undoing it stops being free. This is the column that people skip, and it is the one that changes the character of the others.

Two filled-in rows

Take releasing an order that is on credit hold. What it touches: the hold flag on that order, and the picking queue it drops into. How it is undone: put the hold back. Who can undo it: whoever is covering the credit desk that day. The window: until the picking list prints, because after that the order is physically being assembled and putting the hold back means walking into the warehouse. Four short answers, and the last one is the reason this action can be automated at all: there is a clean point where the cost changes, and it can be written down.

The rows in the third group are shorter and less comfortable. Sending a customer the confirmation that their order has shipped. What it touches: their inbox. How it is undone: it is not. Who can undo it: nobody. The window: there is none. Nothing about that row is difficult to fill in. What is difficult is deciding, once it is written down in front of you, whether anything should be allowed to send it without a person looking.

The finished list is for two things. It tells you which actions can run without anybody watching, because the worst case is a correction someone can make. And for everything else, it tells you what has to be true before it runs: who approves it, and how long they have to change their mind. That second part is why the approval belongs in the same table as the action, with the name and the window as fields: a permission rather than a screen. An approval that lives only in a screen cannot be queried afterwards; what it leaves behind is the order in which things were clicked. If you write nothing else, write the third group. It is short, and it is the only one where being wrong cannot be corrected afterwards.

10. A number you can open

Writing is the dangerous half of the system, and it has taken the previous sections. Reading is the half used every day. Someone asks what no single system can answer alone, and wants a figure they can put in front of a meeting without anyone arguing about it. That calls for two pieces that are not the same. One is a physical place where those tables can finally meet, with its refresh rhythm declared and the age of the data in plain sight. The other is the guarantee that the number coming back was not invented by a language model, but comes out of a calculation you can open and check by adding it up.[22]

The twin

The hard questions of daily work cross systems that have never met: the CRM knows about complaints and the ERP about invoices, and there is no physical place where those tables cross. The twin is that place: synchronised copies of only the necessary tables, each with its cadence declared, because freshness has a price. The report's illustration has the ERP syncing every 30 minutes and the CRM every 24 hours, and the question "unpaid invoices with an open complaint?" answered in one query, with the customer that appears in both, joined.

The connection part is less glamorous than the diagram. When a client's ERP has no API, we put a small computer inside their office with a private connector that exposes agreed queries, which reaches the database locally and feeds the twin through an encrypted tunnel. The twin is read-only for every agent: whatever is wrong in a replica gets fixed in the source system, through an action, never in the copy. Whatever has to be accurate to the second is

not replicated; it is queried live at the source. And when a source stops synchronising, the twin says so: the answer arrives stamped with the age of its data, instead of quietly serving yesterday as though it were now. That stamp has a concrete shape: this balance is from 05:12, the ERP has not synced for three hours. Stale data that declares itself can be used with judgement; stale data served as fresh is a lie told by the system, and a system that lies once never rests free of suspicion again.

Two levels of function and a visible breakdown

Language models read intent well and calculate badly, and it is measured. In 2023 two widely used models answered 55 and 59 per cent of three-digit by three-digit multiplications correctly, and in 2024 adding one clause that changes nothing about the sum cut accuracy by up to 65 per cent across every model tested. Current models score better on those tests; the failure mode has not moved, it appears when the problem grows.[23] So no figure comes out of a model.

A base function reads one concrete thing. A high-level function composes several and returns the finished answer with its breakdown on show. The report's example is available credit: a base function reads the limit granted from the ERP; two base functions, each with its own source, read what is unpaid and outstanding; the high-level function composes the two and shows the subtraction. The client wants to see the subtraction, not a statistic, and that demand is what turns a pretty answer into an auditable one. The difference shows the day someone disputes the figure. With the breakdown in front of them, the conversation is about whether one invoice was posted correctly. Without it, the conversation is about whether the system can be trusted. The first one is settled in ten minutes.

In the public version of the method this becomes two plain sentences that a buyer can hold us to: every business question is a function written by hand, with its formula; and there is no free report generator, because a free report generator produces reports that cannot be compared with each other.[20] It also means no forecasting with trained models: the signals are explicit rules that the person responsible can argue with and change.

11. Three classes of agent, and a ladder

With the layers in place, what runs on top are agents. To decide whether they are trustworthy, the model inside them is the wrong question: it changes every few months. The deciding questions are three others: how much they can break, what they had to prove before being switched on, and what stops them. That is why agents are classified here not by how clever they look but by what they risk, and they climb a step with the evidence of the step below.[24]

An agent that *answers* (chat or voice over the model and the documents, citing its source) risks a wrong answer. An agent that *watches* (fires on a clock or an event, and opens a case or an alert) risks one warning too many, or one too few. An agent that *acts* (runs work end to end, stopping to ask for a signature where a rule requires one) risks a real write. An agent that only watches and warns is switched on the first day. One that writes into the client's system arrives last, and arrives with a frozen baseline behind it so that someone can say whether anything improved.

Before acting for real, an agent goes through shadow: it works for weeks alongside the person, preparing everything and executing nothing, and the comparison decides. The budget comes with a visible brake: a warning near the ceiling and a full stop on reaching it. And the levels only move with the client: on the public site, the levels rise only with the client's approval and fall on their own if quality drops.[20]

Vertical agents, built around the vocabulary and the rules of one sector, are where this work pays twice. An agent processing orders for an industrial distributor has to know what a customer-specific price agreement is; one for an insurer, what risk appetite means. That sector knowledge is the barrier, and we keep it: every sector we model leaves a base the next project starts from, and the second company in a sector does not start from zero. It starts from the first one's map, anonymised.

12. What it runs on today

None of the above is a plan. At the time the report was written, three of these systems were running and one was open to the public, each on a different rung of that ladder and each with its limit published. A method with its scope written down is a method; a method without one is a hope.[25]

The system, and what it does	The declared limit
Logistics, in production: 24 rules over live warehouse and ERP data, with over 30 million operations and 460,000 locations	Detects, explains and warns; never moves an order
B2B sales, in production: 29,000 customers recalculated nightly over 14 million sales lines	Prioritises and warns; the salesperson decides
Support, in deployment: reads the mail, builds the file against the ERP and leaves the reply drafted	Nothing goes out without the approved autonomy level
Retail, in public use: answers on catalogue and terms and starts defined procedures against the ERP	Only the procedures the company has defined

The second case shows why this belongs in the model rather than in the query behind a report: the buying rhythm is each customer's own, measured over their last two years, and one threshold for everyone would have called half the file dormant.

Those are counts of what the systems run over, not of what they improved, and the report is my own deposit rather than an audit by anyone else. So they are a description of size and nothing more. The measured results that Hellomatik does publish are published case by case on its own site, each with what it does not show, and I will quote a few of them here in the form the cases use, without naming the installation.[26]

One warehouse, in numbers

In an automated warehouse we measured, an order takes three minutes to pick. Getting the order to the person who picks it takes four and a half hours, at the median. Nobody had ever measured that wait, because it lived between two systems. That figure is a measurement, not a result: it has no "after", and the case says so. What it does say is that the three minutes of picking are not where the time goes, and the queue at the entrance is.

In the same installation, 41 per cent of the emails in the customer-service inbox were people asking where their order was. Now, every morning, a list arrives of what did not ship yesterday, with the cause. Whether that percentage falls is measured from the day the channel opens, and the channel opens when the client signs; there is no "after" for that figure yet either, and I am not going to invent one.

Closing a year of transport invoices, 180 a month in 30 formats, takes eight minutes; it used to take days of work by a person who knows what they are doing. That one is a result, and it is quoted with its before. And 2,132 orders were held with no reason recorded anywhere. That is not a failure of the people. It is what happens when the map is in their heads and not on paper, and it is the group that has to be investigated first, not the group that is in order.

Every figure Hellomatik publishes comes from a case measured at a customer's site, and every case is published with what it does not show. If we cannot name the customer, we say "an automated warehouse" or "a distributor that sells to other businesses" and keep the number. The number is the proof; the logo is not.[3]

13. When a dictionary is enough

Many of the companies that ask for agents need the cheap piece first, and saying so is part of the first conversation. If the pain is that two dashboards disagree about last quarter, a

semantic layer closes the case with far less machinery. The two things are contracts over the same data, and they are different contracts.[27]

A semantic layer gives one definition of each metric, shared by every tool. It is read-only by design. It serves people looking at dashboards, and a wrong number costs a meeting. An ontology holds entities, relationships, functions and typed actions. It reads and writes, with a human signature where a rule requires it. It serves agents that execute work, and a wrong change would cost money, hence the governance. The boundary is whether the system can change anything, and who authorises it.

The deciding question is a different one from the one buyers usually ask. It is not which is better. It is: should the software prepare or execute any of the decisions the company makes and carries out every day? If the answer is yes for even one of them, reporting tools fall short, however good the metric definitions are.

There is a fair objection to that boundary, and well-informed clients make it: semantic layer vendors are adding agents and write capabilities, so the line moves every year. It does move. What survives is the checklist: when a semantic layer gains typed actions, deterministic functions, human signatures and an auditable log, it has become an ontology, whatever its pricing page calls it. The name matters far less than the list.

The first exercise we do with every client costs one afternoon and zero software: write down the five decisions the company executes every day and, next to each one, how it gets done today. That list, in plain language, is the first draft of the map.

14. How it starts inside a company

Nothing here has to be finished to be worth having, and that is the part clients believe last. The order we follow is designed so that each step pays for itself before the next one begins.[28]

It starts with one source connected and profiled. That alone answers questions that today cost someone an afternoon in a spreadsheet, and it happens before a single entity has been named. Then the model gets written around the first agents, ten to twenty-five entities, and the same questions start being answered with the company’s own words instead of table names. The first agent only answers, citing where each figure came from. Nobody has signed anything yet, and nothing has been written anywhere. Before any agent writes, the current cost of the task is measured and frozen, and the agent spends weeks in shadow next to the person who does it today. Only then does the first typed action get switched on, with its approver, its undo window and its budget.

At every one of those steps the model is exportable and documented. If we stop on step two, the company keeps the map of itself that it did not have before, and that map is worth having whether we build the agents or somebody else does.

The one requirement to start is not technical. Somebody inside the company has to be able to say, for each decision under discussion, what the right answer is and why, because that is what the model writes down. If that person does not exist or has no time, the project stops there, and it is far better to learn it on day one than in month three.

On Hellomatik's site this is called the first cycle, and I use that name and no other: not a pilot, not a trial. The entry is a cycle with the company's own people and real data, and at the end of it there is a task of theirs working and measured. Before an agent is switched on there are three steps the engine forces: a frozen baseline, measured with the client's team, of what the same task cost by hand; at least one written stop rule, saying which cases require their signature; and a rehearsal, first against the history and then in parallel with the real operation. After that, every two weeks inside with the new version. That is not support; it is how we find out what to build next. The first cycle is paid for. The licence is for the whole company, with unlimited users, and the figure is agreed in the first meeting and not published.[29]

15. The cliff, and the last person

There is a point in building something new where the hard part changes, and I have a name for it: the cliff of innovation. It is the point where what you want to make is either not invented yet, or has to be assembled from so many different parts that the difficulty stops being building it. The difficulty becomes integrating it: putting the parts together well enough that what you have looks like one ecosystem, and not like a Frankenstein.[30]

When Hellomatik began, the easy road was clear. Take one of the tools that already existed, add our own touch, and sell it. Thousands of companies did exactly that in 2023 and 2024, and there is nothing wrong with the choice: it is fast, and the risk is known in advance. We went the other way, and the reason was not a taste for difficulty. It was the hands. To do the thing, the system has to know which things exist in that business, which rules the business already follows, what can be undone if it goes wrong, and who is allowed to say yes. Each time we tried to take a step forward and see how far a tool like ours could actually go, we got a little closer to the cliff.

Take an honest inventory of what a system that acts inside a company is made of, and almost every part turns out to exist already, and to be good. A model that understands speech, a model that understands language, a database, a phone line, a calendar, a mailbox,

the invoicing program, the warehouse system, the customer list. Nobody starting today has to build any of them. What does not exist, for any given company, is the thing that makes those parts behave like one system that knows that company: what is true, which rules it already follows, which actions can be undone, who is allowed to approve what. Those four things are the map. They are not parts you can buy. They are the integration, and there is no version of them that can be built once and sold to everybody, because each company's truth, rules, reversibility and permissions are its own.

You can tell a badly integrated system from across the room, but it is more useful to say what you are seeing. Two definitions of the same thing, so that the person using it learns which answer to trust in which screen and carries that around in their head. The person as the translator, copying a reference from one window into another; count the copies and pastes in an ordinary morning, and each one is a seam the system left open and a person is holding shut with their hands. Nobody able to say in one sentence what happens if the system does something and it is wrong. Approval living in somebody's inbox, where the system cannot see it and cannot ask for it. And, most common of all in 2025 and 2026, a chat window as the front door: when the parts do not agree, the easiest thing to put in front of them is a box for text and a model that answers, and it feels like integration and is the opposite, because the model is now the translator and the person is still checking.

Why the year goes there

I want to say why this work takes the time it takes, because from outside it can look like a company that has stopped building. The first reason is old. Fred Brooks, in 1987, separated the essential difficulties of software from the accidental ones, and argued that the essential ones, the ones that come from what the software has to represent, would not yield to better tools.[31] The four missing things are essential in exactly his sense. No model, however good, can tell you which of a company's three addresses is the true one, because that is not a fact about language; it is a fact about that company, and somebody there has to decide it. The second reason is that most of what has to be written down is tacit. Michael Polanyi's observation that we know more than we can tell is the daily condition of a small company.[32] The rule is real, it is followed every day, and nobody can recite it until a system breaks it; so the integration cannot be done from a desk. The third reason is structural: Herbert Simon described in 1962 how complex systems that survive tend to be built from parts that are nearly, but not completely, independent of each other, and the seams above are exactly where that "nearly" lives.[33] The fourth is Conway's law: a system's structure copies the structure of the organisation that built it, so the seams in a company's software are usually the seams between its departments, and closing them often means somebody in

the company agreeing, for the first time, on which department owns which fact.[34] That is not a technical conversation, and it cannot be had by a supplier alone.

The last person

Standing at the cliff, you also find out what a tool like this is really limited by, and it was not what I expected. It is not the model, and it is not the computing power; both get better every month without any help from us. I said it once without planning to, and it has stayed true since: the possibilities end where the creativity of the last person who uses it ends.[30]

A system that can act inside a company can only do what somebody inside that company thought to ask of it. The parts can be excellent, the integration can be complete, every rule can be written down, and the system will still sit there doing the twelve things somebody asked for on the first day, because nobody asked for the thirteenth. The ceiling of the deployment is not in the software. It is in the number of people in the building who look at a piece of their own work and think: this could be asked.

That leaves two jobs. The product's job is to make asking easy: not to make the system clever, which the models do on their own, but to remove every reason a person might have for not asking. The fear that it will do something that cannot be undone is answered by writing reversibility beside every action. The fear that it will do something they were not allowed to authorise is answered by the permission table. The friction of having to describe the company before describing the request is answered by the map, which already knows. The company's job is to have people who ask, and that is the part no supplier can deliver. The people who raise the ceiling of a system like this are not the ones with the best grades or the right title. They are the ones with ingenuity rather than stored knowledge: people who can do a task that did not exist last year with a tool that did not exist last month, and who do not need to be told that it is allowed to ask. I look for two signs in them, neither of which appears on a CV, and I have written about them elsewhere.[35]

16. What the map is for

I should say what all of this is in aid of, because a map is a means, and I wrote down the end in November 2025, before most of the above was published.[36]

It is reasonable to assume that, at some point in the evolution of technology, running a company will stop being a manual, fragmented activity spread across many separate tools. The historical pattern has always been the same: when a task is repetitive, can be expressed as rules and is necessary for an organisation to keep going, sooner or later it gets automated. The machine of the future, as I called it then, is the tool that can take actions in the real world when you write what you need, in plain language. Its only job will be to

operate. If the whole management of a business can be concentrated in one platform, reachable from anywhere and able to carry out tasks on its own initiative, the barrier to starting a company drops sharply, and the number of independents rises: not because of a cultural change, but a structural one. When costs fall, participation rises.

What stays human is the deciding. The value of a company is the people who work in it; that will always be true, however far the technology goes. What is also true is that a company's clients value it not for its headcount but for its ability to meet their needs. So the machine of the future is the one that lets a legal person depend a little less on physical persons for its existence, and all of it works only with a human component in its decisions. Most people, when they build an AI system, see it as a program that is a bit easier to operate. That is the wrong layer. Its real power is in giving it a purpose and rules for making decisions, and the map is where those rules live.

The map is also the answer to the fear that goes with that picture. People ask what the goal looks like, and the honest answer is that the picture in my head is HAL 9000: a system that understands the whole company and can act inside it. I say that with some humility, because HAL is a cautionary tale, and the reason it is one is the same reason our work is mostly about limits. A system that understands everything and answers to nobody is the failure mode, not the goal. That is why every action carries whether it can be undone and whose approval it needs, and why the permission table is the part I would show a sceptic first.[4]

And there is a smaller, more personal version of the same end, which is the one I actually work for. A company, to me, is like a child: if you look after it, it grows, and it needs you every day. Anyone who owns a small business knows that feeling: the wish to step away for a few days and come back to find the clients answered and the work done. What the map makes possible is that the child grows on its own. Put without the metaphor: the difference between having a business and being tied to it.[36]

That is why, when Hellomatik takes on a company, the first thing we write down is not what the system will do. It is what the business already does: the rules it follows, the actions it takes, who approves what. The agents are then given a goal and an intention, not a chat window. They are aimed at a job that a specific person in that company already has, and their success is measured by whether that person's afternoon got easier. A product is aimed at one specific person, or it is aimed at nobody; I learned that selling, not building, and it holds for agents exactly as it held for dashboards.[37]

17. What it costs, and what I cannot show

The cost of writing the map falls on the company rather than on me, and it should be said plainly. It puts people in a meeting arguing about something they had been quietly disagreeing about for years. The count of live exceptions comes out higher than expected. And writing the rule down does not settle who is allowed to change it once it is written, which is the failure mode the older discipline of data governance is full of and which I do not consider solved.[6] Add the costs the rules impose, which I listed in section 7: no free report generator, no trained-model forecasting, a nightly load that takes hours, an agent that takes time to switch on, daily series that reach yesterday.

Then there is what this essay cannot show, and I would rather list it than have it found. Everything above is design and method, and three concrete limits bound what can be concluded from it. The measured figures in the report come from third parties, not from our own deployments; we do not publish numbers we have not verified end to end. The bias has a direction: we design and sell systems of this kind, so the report probably overrates actions and underrates how far a well-run semantic layer stretches. And the line between the two rests partly on vendor definitions, which change faster than the architecture; the distinction survives, the commercial names do not always.[38]

Beyond the report's own limits: there is no controlled comparison anywhere in this. I have not built the same system both ways and measured the difference. The four missing things are the ones we found; other companies at the same cliff may find a fifth. The working paper on the cliff is self-published and not peer reviewed, and it cites my own work three times in seven references; those weaknesses compound rather than cancel. Very little of what I have written can be verified from outside. Hellomatik's first client was a large clinic, and I am not naming it or its city, so that is not a case study and I am not going to present it as one; it is a fact about my working life that a reader would have to take on trust, which is the same footing as most of this essay. The DOIs are addresses rather than reviews: they fix the argument in one place with a date on it so it can be disagreed with in detail. And I am a co-founder of the company whose method this describes, which is a conflict of interest the reader should keep in view.[5]

18. Three predictions, dated

A description that predicts nothing is a story. These three predictions come from the working paper on the cliff, and I repeat them here with their checks, dated so that being wrong on one of them is visible rather than forgotten.[39]

First. Products that took the easy road, an existing tool plus a touch, will converge on each other and compete on price, because the parts they share are the same parts and the touch is small. Check: compare the feature lists and prices of such products in 2028 with those of 2025. If the lists have converged and the prices have fallen, the prediction held.

Second. This one favours my own company, and a reader should weight it accordingly. The durable value in this category will belong to whoever holds a company's written map: its truth, rules, reversibility and permissions. Because those four things are specific to each company and cannot be bought, the party that has written them down will be hard to replace even when the models underneath are swapped. Check: in 2028, ask companies that changed their AI supplier what they kept. If what they kept was the written account of how they decide, the prediction held.

Third. The ceiling of a deployment will correlate with the number of people in the company who ask the system for new things, more than with the model, the budget or the size of the company. Check: for any set of deployments, count requests per person per month against the number of distinct tasks automated. If the first explains the second better than the model version does, the prediction held.

The report, the working papers and this site all carry dates and identifiers precisely so that these checks can be made against what was actually claimed, not against a memory of it. The three, with one earlier bet, are kept in one place, with their dates and their status.

19. In one paragraph

A company already has a map of itself. It is written in the heads of the people who reconcile things by hand, it disagrees with itself from one department to the next, and it has never been on paper because until now nothing needed to read it. Software that acts needs to read it. So the work is to write it down: which system is believed about each thing, which rules the business already applies and where they stop, which actions can be undone and how, and who may approve the rest, recorded as data with a name and a date on it. Everything that acts sits on top of that and cannot go around it. The model inside the agent is a commodity and will be swapped; the map is the part that is the company's own. Build it backwards from the first agent, so that it is worth having before it is finished. Let the agents climb from answering to watching to acting, each step with the evidence of the step below. Publish the limits with the method, and count what it costs. And remember that the ceiling was never in the software: it is in how many people in the building think to ask.

Isaac Correa, Madrid, September 2026. ORCID 0009-0003-0657-4419.

Notes

1. The documents this essay is drawn from: Correa, I. (2026), *How we build a company's ontology: six layers, twelve rules and one procedure*, technical report, Hellomatik, Madrid, August 2026, DOI 10.5281/zenodo.22647735; Correa, I. (2025), *The machine of the future*, working paper, DOI 10.5281/zenodo.22671771; Correa, I. (2026), *The cliff of innovation*, working paper, DOI 10.5281/zenodo.22704763; and the pages of this site cited below.
2. What selling taught me that building never did, on this site.
3. The account of the person who reconciles by hand, and the rule about naming a customer, are the ones Hellomatik uses in public; the cases themselves are at hellomatik.com/use-cases.
4. How I work on a product, on this site.
5. The company's registration is described, with the BORME reference, in How Hellomatik started; Kodalogic is described in Copy until you find creativity where there is none.
6. How Hellomatik started, on this site: the restaurant voice, the two cheap fixes, the older discipline, the document, and the hands.
7. The definition, and the reasons for keeping the word, are in the same page and in the opening section of the technical report (note 1), "What everyone else sells when it sounds similar".
8. Your company already has rules. Nobody ever wrote them down, on this site.
9. Four questions to ask before automating anything, on this site, where the questions can be worked through.
10. The vocabulary I work in, on this site.
11. The three products are named in the opening section of the technical report (note 1), which cites their own documentation: dbt Labs, *The dbt Semantic Layer*; Snowflake, *Semantic views*; Microsoft, *Power BI semantic models*.
12. Technical report (note 1), part 1, "The six layers".
13. Technical report (note 1), "We model backwards from the agents".
14. Rei Labs, *Emergence: autonomous structure discovery*, 2026, as cited in the technical report; the figures are the authors' own, self-reported.
15. Technical report (note 1), "Three mistakes of ours, and what they changed".
16. Technical report (note 1), part 2, "The nevers and the always".
17. Gartner, 2025, on agentic AI projects cancelled before the end of 2027, as cited in the technical report.
18. OWASP GenAI Security Project, *Top 10 for LLM Applications*, 2025 and 2026 editions, as cited in the technical report.
19. Technical report (note 1), "What a typed action carries declared" and "The journey stops at the signature".
20. The seventeen rules, with what they cost, at hellomatik.com/rules; the levels of agent and the first cycle on the same site.
21. Which actions can be undone, on this site.
22. Technical report (note 1), part 3, "The reading side": the twin, the connection and the functions.
23. Dziri et al., *Faith and Fate*, NeurIPS 2023; Mirzadeh et al., *GSM-Symbolic*, 2024; both as cited in the technical report.
24. Technical report (note 1), part 4, "Three classes and a ladder".

25. Technical report (note 1), “Four of our systems running today, with their limits”. The table is reproduced as it appears there.
26. The warehouse figures are from the published cases at hellomatik.com/use-cases: the inbound queue, what did not ship yesterday, the transport invoices, and the anti-patterns page. Each case states what it does not show.
27. Technical report (note 1), “When a semantic layer is enough” and “The boundary moves; the checklist survives”.
28. Technical report (note 1), “How this starts inside a company”.
29. hellomatik.com/first-cycle.
30. Correa, I. (2026), *The cliff of innovation: why integration, not construction, is the hard part of building a system that acts inside a company*, working paper, DOI 10.5281/zenodo.22704763; the short version is on this site.
31. Brooks, F. P. (1987). No silver bullet: essence and accidents of software engineering. *IEEE Computer*, 20(4), 10–19.
32. Polanyi, M. (1966). *The tacit dimension*. University of Chicago Press.
33. Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6), 467–482.
34. Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28–31.
35. The people I look for, on this site.
36. Correa, I. (2025), *The machine of the future: what the next tool is, and what it will always need from us*, working paper, November 2025, DOI 10.5281/zenodo.22671771; also on this site.
37. A product is aimed at one person, or at nobody, on this site.
38. Technical report (note 1), the closing section on its limits.
39. The cliff of innovation (note 30), section 8.

About the author. Isaac Correa co-founded Hellomatik in Madrid on 28 July 2025, and co-founded Kodalogic in September 2024. He is a co-founder of the company whose method this essay describes, which is a conflict of interest the reader should keep in view. The method itself is published as a technical report with a DOI: 10.5281/zenodo.22647735.

Sources and full record: isaac-correa.github.io · ORCID 0009-0003-0657-4419 · Licence: CC BY 4.0