The cliff of innovation

Why integration, not construction, is the hard part of building a system that acts inside a company — and where such a system's possibilities really end

Isaac Correa · Co-founder, Hellomatik · Madrid, Spain
Hellomatik working paper, Madrid · Self-published, not peer reviewed · September 2026
ORCID 0009-0003-0657-4419 · A shorter version appeared on the author's site, isaac-correa.github.io/cliff

ABSTRACT

This paper names and describes a point that the author reached while building Hellomatik, a Madrid company whose product acts inside other companies: 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 and becomes integrating it well enough that the result behaves like one system rather than a collection of parts. The paper sets out why most of the parts of such a system already exist, what does not exist for any given company (a written account of what is true, which rules the business already follows, which actions can be undone and who may approve what), how a badly integrated system can be recognised from the chair of the person using it, and why the integration is where the effort goes. It then states the limit the author found at the edge: the possibilities of a system that acts inside a company end where the creativity of the last person who uses it ends. Three checkable predictions follow, along with the paper's limits.

1. Two roads

When Hellomatik began, in the summer of 2025, the easy road was clear and I want to describe it fairly, because a lot of good companies took it. You take one of the tools that already exist, add your own touch, and sell it. Thousands of companies did exactly that in 2023 and 2024, once language models had become something you could rent by the call. Most of what they built had the same shape: a form, a box for text, and somebody else's model behind it. I name none of them on purpose: the point is the shape, not any one company, and several of them are good businesses. There is nothing wrong with the choice. It is fast, the risk is known in advance, and the customer understands what they are buying.

We went the other way, and the reason was not a taste for difficulty. Our starting point was small: a system to answer the phone in a restaurant, with a voice that takes the call, a brain that decides what the call means, and a pair of hands that does the thing. The hands were the problem. 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 a place I started calling the cliff.

This paper is about that place. It is written from one company's experience, in the first person, and it makes no claim to have discovered something nobody else has seen. What I think is useful is the name, the description of what you find there, and the limit at the edge, because the three of them changed what we spent our time on.

2. The definition, taken apart

> The cliff of innovation 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.

There are three claims in that sentence and they are worth separating.

The first is that there are two ways to be at the edge. One is that the thing does not exist yet: nobody has made it, and you will have to. The other is that all of its parts exist, but nobody has made them into one thing. In 2025 the second case was far more common than the first, and it is the case this paper is about. The interesting difficulty of our decade is not that the parts are missing. It is that they are all there, made by different people for different purposes, and they do not know about each other.

The second claim is that, at the cliff, the difficulty changes kind. Below the cliff, the hard part of a software product is building it: writing the code, getting it to run, getting it to run fast. At the cliff, the code is not where the days go. The days go into making parts agree: about what a customer is, about what happened at three in the afternoon, about whether an order that has been confirmed can still be cancelled and by whom. That work does not look like programming from the outside, and it is easy to mistake it for delay.

The third claim is the test. An ecosystem is a set of parts that behave as one thing from the point of view of the person using them. A Frankenstein is a set of parts that have been attached to each other, and it shows. The difference is not aesthetic. It is measured in what the person using the system has to hold in their head, and section 4 sets out how to see it.

3. What exists, and what does not

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 well enough to take a phone call exists. A model that understands language well enough to read an email and say what it asks for exists. Databases exist and have for decades. The phone line, the calendar, the mailbox, the invoicing program, the warehouse system and the customer list all exist, and in most companies they have existed for years. Nobody starting today has to build any of them, and anyone who does is spending money to arrive where everybody else already is.

What does not exist, for any given company, is the thing that makes those parts behave like one system that knows that company. Concretely, four things are missing, and they are missing in every company we have worked with, not because the companies are careless but because nobody had ever needed them written down.

What is true. A company has several systems and each of them holds a version of the same fact. The customer's address is in the invoicing program, in the delivery system and in a spreadsheet, and on any given Tuesday they disagree. Before a machine can act, somebody has to decide which system tells the truth about each thing. Not in general: thing by thing.

Which rules the business already follows. Every company runs on rules, and almost none of them are written anywhere. An order over a certain amount waits for a signature. A particular customer is never called before ten. Deliveries to one postcode go with one carrier because the other one lost a pallet in 2023. The people who work there know these rules the way they know the way home, and they will tell you them only when a system breaks one.

Which actions can be undone. Sending an email cannot be undone. Reserving stock can. Issuing an invoice can be reversed, but only through a second document and only within a period. A system that acts has to know, for each action it might take, which of these three it is, because the answer decides how much checking has to happen before the action and not after.

Who may approve what. Some actions are anyone's to take. Some need a named person to say yes. Some need two. This table exists in every company and is written down in none of them, and it is the part I would show a sceptic first, because it is where a system that acts is either safe or is not. So that this claim can be checked without leaving the paper, here is the shape of the table, with an illustrative row of each kind; the rows are invented for the example and describe no client:

Action	Can it be undone?	Who has to say yes	What the system does
Reserve stock for an order	Yes, by releasing it	Nobody	Acts, and records it

Send a quotation to a customer	No; once sent it is read	The account owner	Prepares it, asks, then sends
Issue a credit note	Only through a second document	Finance, and the account owner	Prepares it, asks both, then issues

Every row has the same four cells. If a company cannot fill the second and third cell for an action, the system does not take that action; that rule is the whole of what makes it safe to let it act at all.

These four things are what we came to call a company's ontology, and the way we write them down is published as a technical report with a DOI [1]. I am not going to repeat the method here. The point for this paper is narrower: the four things are not parts you can buy. They are the integration. They are what makes the parts you can buy behave like one system that knows one company, 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.

4. How you recognise a Frankenstein

You can tell a Frankenstein from across the room, but it is more useful to say what you are seeing. Here are five seams, each of them observable from the chair of the person who uses the system, and each of them something we have had to remove from our own work at least once.

Two definitions of the same thing

Ask the system what a customer is and you get two answers, because the invoicing part and the delivery part were made by different people. The person using the system learns which answer to trust in which screen, and carries that knowledge around in their head. The moment they leave the company, the knowledge leaves with them.

The person as the translator

Every seam that the parts do not close, a person closes. They copy a reference from one window into another. They know that the status "sent" in one part means "printed" in another. A good test of a Frankenstein is to count the copies and pastes in an ordinary morning: each one is a seam the system left open and a person is holding shut with their hands.

Nobody knows what can be undone

In a Frankenstein, the question “if the system does this and it is wrong, what happens?” has no answer that anyone can give in one sentence. So either everything is checked by a person before it happens, which removes most of the point, or nothing is, which removes the company’s trust the first time something goes wrong.

Approval lives in somebody’s inbox

The permission to do a thing is not in the system; it is in the fact that one person forwards the request to another and waits. The system cannot see the approval and cannot ask for it, so it cannot act on anything that needs one, which in most companies is most of the things worth acting on.

A chat window as the front door

The last seam is the most common one in 2025 and 2026. 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. It feels like integration and it is the opposite: the model is now the translator, guessing what each part meant, and the person is still checking. A box for text in front of somebody else’s model is not a product. It is a toll booth.

An ecosystem, from the same chair, is the reverse of all five. There is one definition of a customer and every part uses it. Nothing is copied by hand, because nothing needs translating. Every action the system can take carries, written beside it, whether it can be undone and who has to say yes. And the person using it does not talk to the parts at all. They say what they want, and the system already knows enough about their company to do it or to ask the right person first.

5. Why the integration is where the year goes

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 a 1987 paper, 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 [2]. The four missing things in section 3 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 [3] is the daily condition of a small company. The rule about the carrier and the postcode 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. It is done by putting a first version in front of the people who hold the rules, watching it break one, and writing the rule down. That takes calendar time, and no amount of engineering shortens it.

The third reason is structural, and Herbert Simon described it in 1962 [4]: complex systems that survive tend to be built from parts that are nearly, but not completely, independent of each other. The seams in section 4 are exactly where that "nearly" lives. The parts of a company's software were bought separately because they are nearly independent; the work of integration is finding the small number of places where they are not, and closing those and only those. Closing seams that were never open is wasted work; leaving open the ones that matter is a Frankenstein.

The fourth reason is Conway's law [5]: a system's structure copies the structure of the organisation that built it. The seams in a company's software are usually the seams between its departments, and closing them in software often means somebody in the company agreeing, for the first time, on which department owns which fact. That is not a technical conversation, and it cannot be had by a supplier alone.

Put together, these four reasons are why we chose to publish the method as a report rather than as a list of features. Features are the parts. The method is the integration, written down so that somebody else can check it, and so that a company can see what it is agreeing to before any software is installed.

6. What the method is running on

A method with its scope written down is a method; a method without one is a hope. The technical report [1] states the scope of what the approach is running on rather than implying it, and I repeat it here because it is the only evidence in this paper that is not an argument.

System	What it does	Scope, as stated in the report
Logistics	Rules over live warehouse and ERP data	24 rules written down; over 30 million operations; 460,000 locations in the warehouse system
B2B sales	Nightly recalculation over sales lines	29,000 customers; 14 million sales lines

Two further systems	In production, within the same method	Described in the report

None of these figures is a claim about results. They are a statement of where the integration described above has been done, on what volume of data, and with how many rules written down. Anyone who wants to check the claim can read the report, which carries a DOI and does not change.

7. The last person

Standing at the cliff, you also find out what a system like this is really limited by, and it was not what I expected.

It is not the model. The models get better every month without any help from us, and a system that is built around the four things in section 3 gets that improvement for free. It is not the computing power, for the same reason. The limit I found is this:

> The possibilities end where the creativity of the last person who uses it ends.

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, and I want to be precise about whose they are.

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 (answered by writing reversibility beside every action), the fear that it will do something they were not allowed to authorise (answered by the permission table), and the friction of having to describe the company before describing the request (answered by the ontology, which already knows). I have argued elsewhere that such a system also needs a goal and principles to decide by, not only rules [7]; that argument is separate from this one, and this paper stands without it.

The company's job is to have people who ask. This is the part that no supplier can deliver, and it is why, at Hellomatik, we care as much about who we work with as about what we build. 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 have written elsewhere about how we look for them [6]. Here it is enough to say that the last person is real, that they set the limit, and that the limit moves every time one more person in the building decides to ask.

8. What this predicts, and how to check it

A description that predicts nothing is a story. Here are three things the argument above predicts, each with a way to check it later. They are dated so that they can be checked, and so that being wrong on one of them is visible rather than forgotten.

Prediction 1. 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.

Prediction 2. This prediction favours the author's own company, and a reader should weight it accordingly. The durable value in this category will belong to whoever holds a company's written ontology: 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.

Prediction 3. 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 [1], this paper and the author's 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.

9. Limits of this paper

This is one company's experience, written by one of its founders, and it should be read as that. There is no controlled comparison in it: I have not built the same system both ways and measured the difference. The figures in section 6 are the scope of what the method

runs on, not a measure of its results. The four missing things in section 3 are the ones we found; other companies at the same cliff may find a fifth. And the paper is self-published, not peer reviewed, and cites the author's own work three times in seven references; those weaknesses compound rather than cancel, and the reader should treat the argument as one practitioner's account rather than as an established result. That is why it carries the checks in section 8 rather than asking to be believed.

What I would defend without qualification is the name. Having a word for the place where the difficulty changes kind let us stop treating integration as delay and start treating it as the work.

References

1. Correa, I. (2026). *How we build a company's ontology: data, rules, actions and permissions*. Technical report, Hellomatik, Madrid. DOI 10.5281/zenodo.22647735.
2. Brooks, F. P. (1987). No silver bullet: essence and accidents of software engineering. *IEEE Computer*, 20(4), 10–19.
3. Polanyi, M. (1966). *The tacit dimension*. University of Chicago Press.
4. Simon, H. A. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6), 467–482.
5. Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28–31.
6. Correa, I. (2026). The people I look for. isaac-correa.github.io/people.
7. Correa, I. (2026). *The machine of the future: what the next tool is, and what it will always need from us*. Hellomatik working paper, Madrid. DOI 10.5281/zenodo.22671771.

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 paper 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