A STORY AND A METHOD

WITHOUT EVER LEAVING

Building an engineering business that sells abroad, from Algeria, where nobody goes

LEGACY SYSTEMS · CLOUD · PRACTICAL AI
PROSPECTING · NEARSHORING · SOVEREIGNTY

ADAM BENBRAHIM
Information systems engineer

FIRST EDITION · ALGIERS, 1 AUGUST 2026

A STORY AND A METHOD

Without Ever Leaving

Building an engineering business that sells abroad, from Algeria, where nobody goes

ADAM BENBRAHIM

First edition · Algiers, 1 August 2026

Without Ever Leaving
Building an engineering business that sells abroad, from Algeria, where nobody goes

First edition. Completed in Algiers, 1 August 2026.

This book is part personal account, part working manual. It follows a real path: Algerian state schooling, an engineering school, IT production work in a bank, a technical lead role in outsourced services for foreign clients, then a consultancy registered in the United Kingdom and run from Algiers.

No client, employer, partner, colleague, software vendor or product is named. People appear by their role and, where it helps, by their country. The situations are real, but stripped of anything that could identify an organisation. Durations and periods are deliberately imprecise. The figures given are what I observed in my own work, not market statistics.

The author wishes to record his gratitude to the institutions and teams that trained and hosted him. The analyses of economic or organisational mechanisms in these pages concern general models, widely documented in the professional literature, and target no organisation in particular.

This book accuses nobody. It describes mechanisms. A mechanism has no lawyer.

 Applying the methods described here, however, is strongly encouraged, including against the author.

CONTENTS

What you are about to read

PROLOGUE

A million lines in forty eight hours

> A job sold as several weeks. Delivered in forty eight hours. And the discovery that the real problem was not technical at all.

There is a precise moment when I understood that the rules of the game had changed, and that almost nobody around me had noticed.

A business application. A little under a million lines of code, written in a proprietary language that modern tools cannot read. No documentation. The original developers long gone. The company ran on it, invoiced through it, paid its salaries because of it. Nobody inside could explain in detail how it worked.

The request was simple to state and brutal to carry out: document what exists. Map the modules, the dependencies, the business rules, the special cases. The quote said several weeks. That was honest, and by industry standards it was even optimistic.

I spent the first morning looking for a way in. The code could not be opened with a normal editor, the proprietary environment was compulsory. Digging through the menus, I found an export function. It produced a document of more than ten thousand pages containing all the source code in plain text.

From that point it was no longer a language problem, it was a plumbing problem. I wrote a script that split the document, found the separators, rebuilt each unit of code and grouped them into coherent batches. I set up a chain that sent each batch to a language model with a strict instruction: translate nothing, rewrite nothing,

describe. What data goes in, what comes out, which rules apply, which special cases are hard coded, which dependencies exist with the other units.

Forty eight hours later I had the documentation. Not a surface summary: a usable map, module by module, with the business rules pulled out and the risky areas flagged.

Several weeks sold. Two days spent.

That was the moment I stopped seeing a tool and started seeing an economic problem.

The four doors

I sat in front of that deliverable and realised I had four options, and that none of them was good.

First door. I tell nobody. I produce believable daily reports for the rest of the calendar and give myself several paid weeks. It is comfortable, it is tempting, and it is the door many people take without ever admitting it. The problem is not moral, it is strategic: you have just bought quiet at the price of your own progress. You will never learn what that discovery was really worth.

Second door. I tell my management. Almost certain reaction: well done, and here are three more projects. With no change in pay, because in a services company you are paid for time, not for outcome. You have increased your output without increasing your perceived value. You have turned a strategic discovery into extra workload.

Third door. I tell the client directly. He is delighted, he pays for two days instead of several weeks, and he immediately asks the next question: if this took two days, why does everything else take months? You have done a favour to one invoice and blown up a business model, including your own.

Fourth door. I tell everyone. This is the door of full honesty and also of complete shipwreck, because it combines the drawbacks of the two previous ones.

I have since put this question to very strong engineers, in training sessions, in interviews, in working sessions. The answers spread across all four doors, and each person defends their choice with good arguments. That is the clearest sign that there is no technical answer. There is only a political one.

> RULE
>
> A radical gain in productivity is never welcomed as good news by an organisation whose revenue is tied to time spent. It is welcomed as a threat. Work out the reaction before you deliver, not after.

What that episode really taught me

I drew four lessons from it, and this book develops them from beginning to end.

One. Scarcity is not where you are told to look for it. Nobody was interested in that application's proprietary language. No student put it on a curriculum vitae. That is exactly why there was a mandate, a budget and no competition.

Two. Assistance from artificial intelligence does not replace an engineer, it multiplies the one who already knows where to look. The model invented nothing. It carried out, at scale, an approach I had designed. The value was in the splitting, not in the generating.

Three. Inside an organisation, a productivity gain that is not captured by the person who creates it is captured by someone else. Always. If you want to keep the value you create, you have to own the structure that invoices it. That is the reason I ended up setting up a company.

Four. The real asset is not the code, it is the knowledge of the system. An organisation that no longer understands its own software does not have an IT problem, it has a survival problem. And it will pay, very well, whoever can explain it without frightening it.

This book

This book explains how you go from that discovery to a business that sells abroad, from a country you never leave.

It talks about state schooling, class rankings, and what a ranking does not buy you. It talks about the engine room of a bank, about legacy systems that keep whole economies running, and about the vendors who own them. It talks about how you set up a company abroad without ever setting foot there, what credibility actually costs, and the banking friction nobody describes to you before you run into it.

It talks about prospecting, seriously and in technical detail, because it is the skill university will never teach you and yet it decides whether twenty years of academic excellence turn into anything at all.

It talks about saying no. Often.

And it talks about staying. I have never filed an application to leave. Not once. That is not ideology, it is a calculation, and I will set it out figure by figure.

Where the crowd is, I am not. Not on principle. On arithmetic.

One last warning before we start.

This book will not sell you a dream. You will find no growth recipe, no magic formula, no promise of passive income. You will find mechanisms, real costs, refusals, mistakes, a few wins, and a repeatable method for going after a foreign market without offering your physical presence as a guarantee.

If you want an inspiring story to tell at a conference, close it now.

If you want to understand how this really works, keep reading.

◆

PART ONE

The ground

Nothing international is built on empty ground. Before we talk about foreign companies, prospecting or markets, we need to understand where the raw material comes from: a state school, a class ranking, an engine room and two years of outsourced work. Five chapters to lay the floor.

CHAPTER 1

A product of the state school system

> Top of the class for twenty years. Then the discovery that a ranking is a currency you cannot spend anywhere.

I am entirely a product of the Algerian state school system. No private school, no foreign lycée, no year abroad. The normal route, the one millions of people follow, from primary school to an engineering degree.

Primary school in central Algiers, top of the class for all five years. Middle school, then another one in the eastern suburbs, with a leaving certificate that put me first in the whole district. Secondary school, mathematics stream, a baccalaureate above seventeen out of twenty, first in the stream in my school. Then the national higher school of computer science, the most selective in the country the year I joined, specialising in information systems. Ranked first out of a hundred and two students in my specialism. Final project defended with the highest honours.

I set these facts out without any particular pride. I set them out because they are the starting point for the only question that matters in this book: what use is any of it, in practice, once you walk out of the school gate?

The honest answer, the one nobody gives you while you are collecting distinctions: far less use than you think.

The currency you cannot convert

A school ranking is a currency. It has real purchasing power, but only inside the territory that issues it.

Inside the school, that currency buys everything. It buys the respect of your teachers, access to the best supervisors, your choice of specialism, the regard of your classmates, sometimes a grant. It shapes your whole identity, because for twenty years you have been told that your rank is the measure of your worth.

Walk through the gate. The currency instantly stops converting.

A recruiter looks at your rank for four seconds. He draws one binary piece of information from it: this person can work and can finish what they start. That is all. It is not nothing, but it is not what you imagined. He will not pay you more. He will not trust you with more. He will not open a door that the twentieth student in the year would not have opened simply by being more relaxed in an interview.

Twenty years optimising a measure that stops being traded on the day you need it most.

I have seen this from both sides. I was the candidate, then I was the person receiving applications, running technical interviews, standing at the stand in recruitment fairs, making the decision. I have read hundreds of files. I have interviewed people from all over.

What I observed is blunt and consistent: academic ranking is almost never the deciding factor. What decides is your ability to

show value that can be used immediately, and the position of strength you take during the conversation.

What school really gives you

Do not misread me. Algerian state education gave me a great deal, and I would not trade it for any degree bought elsewhere.

It gave me rigour. A serious engineering course teaches you to take a problem apart down to the bone, to refuse a vague explanation, to tell an intuition from a proof. That discipline is invisible on a curriculum vitae and decisive in real life. When you open a system nobody understands, that is what does the work.

It gave me self reliance. The system does not hold your hand. There is no personal support service, no careers adviser, no workshop on how to write a curriculum vitae. You sort yourself out. You go and find the information, you set up your own environment, you learn on your own what is not on the syllabus. That reflex is worth a fortune later.

It gave me freedom from debt. This point deserves a moment, because it is the economic base of this whole book. My engineering degree cost me nothing. Zero. No tuition fees, no student loan, no family sacrifice to pay back.

Compare that with the path of an international student in an English speaking country. They pay heavy annual fees, often for several years. They arrive on the market with a debt to repay and a family expectation to honour. That debt then dictates every decision they make. They cannot turn down an offer. They cannot take a risk. They cannot say no to a client. They cannot try something that might fail, because failure has become financially forbidden.

> **RULE**
>
> The first asset of a founder is not capital, it is the absence of debt. A free education buys you room to move that no paid degree can ever give back.

I once spoke with someone who had gone abroad to study. A strong profile, who had methodically built an environment there.

They were looking for part time work in technology. They could not find any.

I was considering offering them a part time contract, properly set up under the rules of that country, to represent my company locally. There was real interest at the start. It did not happen: the people around them felt that working on old technologies would be a step backwards in their career.

Studies paid for at the cost of real effort leave very little room for perceived risk, even when the real risk is zero. Financial pressure ends up deciding instead of the person.

I closed the matter respecting the commitment I had made, and without any bad feeling. The situation was understandable.

Excellence and friction

There is a second truth, less pleasant, that I need to set down now.

Academic excellence has a side effect nobody talks about: it makes you socially unfit for the average workplace.

When you have spent twenty years being judged on whether your answers are right, you arrive in a company thinking that being right is the criterion. You discover that it is secondary.

I would give myself zero out of twenty for emotional intelligence in my early working years. I overcompensated technically. I delivered more, faster, better, thinking that would settle everything. It settled nothing at all. It made me useful and isolated.

An engineer who is right against his environment is wrong. That is a law. You can find it unfair, it still applies.

> **TRAP**
>
> Believing that technical superiority excuses you from political intelligence. It excuses you from nothing. It even makes things worse, because it makes you threatening to anyone whose position depends on the problem staying difficult.

What I drew from that, looking back, fits in one sentence: choose your ground. Inside an organisation you do not own, technical skill alone will never be enough and political intelligence is compulsory. Inside a structure that belongs to you, you can do pure engineering and let the market decide.

I ended up choosing the second option. That is not an ideological choice, it is a way of saving energy.

The maths of leaving

That leaves the question everyone asks and nobody analyses honestly: why stay?

I have never filed an application to leave. Not one form, not one registration. That is not stubbornness. It is a calculation, and here it is, coldly.

Leaving, in the standard pattern, means starting the accumulation again. You arrive with a degree that is not locally recognised, no network, an unstable administrative status, and a permanent need to be validated by an institution. You need a university to validate your right to stay, then an employer for your visa, then an administrative office for your renewal.

I have watched that cycle up close, in brilliant people. A former classmate, excellent, who left with a strong file. He did not find a post matching his level. He took a second degree at the same level as the first. Then he opened a self employed status there, with no clients, purely to keep his right to remain.

I have seen others accept apprenticeship contracts after several years of real experience, because apprenticeships are subsidised and therefore easier to obtain. They came down several rungs to be accepted.

Leaving means trading a position of strength at home for the position of a petitioner somewhere else.

The opposite reasoning, the one I followed, goes like this.

Software work can be done anywhere. The client does not check your address, he checks your deliverable, your legal framework and your ability to reassure him. If I can sell to a foreign market without living there, then I keep the local cost of living, the absence of debt and my family roots.

The reasoning stops there. There is no romance in it. There is an equation.

This book is the manual for that equation. It starts with what school did not teach you, which is exactly what is missing to solve it.

CHAPTER 2

What school will never teach you

> Five years learning how to build. Zero hours learning how to sell. That is the hole through which the value of a whole generation escapes.

There is one gap, only one, that explains most of the waste I see around me.

University teaches you how to build. It does not teach you how to sell.

You leave with real production capacity and no ability at all to turn it into income. You can design a distributed architecture, but you cannot write a message that gets a reply. You can prove the complexity of an algorithm, but you cannot explain to a non technical director why his business is at risk. You can tune a query, but you cannot say no to a client who is costing you money.

The result follows mechanically. Elite engineers graduate and go knocking on doors at cut prices. Some get through, by instinct or luck. The rest stay stuck in roles that use less than a tenth of what they can do.

We train people who can build systems, and who cannot name their price.

The basic reversal

It all starts with a stance. Until that is corrected, no technique works.

The default stance of a new graduate is that of a petitioner. He sends his application, he waits, he hopes, he thanks. He almost apologises for existing. He instinctively uses the words that downgrade him: junior, no experience, I want to learn.

The correct stance is that of a supplier. You do not need a job, you have a service to offer. The conversation is no longer a plea, it is a transaction between two parties assessing each other.

That shift changes everything, including in material terms.

SIX RULES OF STANCE

One. Never use the words junior or no experience. You are shooting yourself in the foot with your own vocabulary.

Two. Never say you have come to learn the trade. A company is not a university. Even if it is true, even if the company trains well, do not say it.

Three. Never beg. If you bring value, you sell it to whoever you choose. Begging puts you below someone when you are the one holding the service.

Four. Be convinced that what you bring is worth more than what you will receive for that role. If you are not, it is the wrong role.

Five. Work in absorption mode, not in a tug of war. At the start you do not talk money first unless you are in a position of strength. You accumulate first, you invoice later.

Six. Rename your document. A curriculum vitae that becomes a service offer changes the nature of the exchange before it is even read.

That last rule looks cosmetic. It is not. A curriculum vitae is an application document, it invites a judgement. A service offer is a commercial document, it invites a buying decision. The same content, in those two forms, does not produce the same conversation.

The starting trio

There is a simple model I ended up writing down after living through it. Three variables: time, money, progress. You cannot

maximise all three at once. Your path is decided by the order in which you choose to sacrifice them.

Phase one. You sacrifice time and money to fill up progress.

You are new. The market is crowded at the entrance. To get out of the mass, you do what others do not: you put time where nobody puts any, you take initiatives nobody asked for, you go beyond your remit. You do not talk about money. You treat this period as a second internship.

It is thankless and it is necessary. Without this phase there is nothing to trade later.

Phase two. Progress is full, money arrives.

You are no longer in the first category. There are fewer people at your level. Experienced people are more sought after because they are productive on day one. You can stop here. Many people stop here.

That is a trap. You have the money for your time, and you stop growing. A few years later someone hungrier replaces you.

Phase three. You must free up time at any cost.

You keep sacrificing time even though you already have the money, but you sacrifice it differently: you build visibility, you launch initiatives, you bring others up with you. You delegate what is settled. You build on what already exists.

The final goal is not to earn more. It is to reduce the time spent, which mechanically increases both money and progress, and incidentally your impact.

The point is not to work a lot. The point is to reach the moment when your time no longer sets your income.

Crowd behaviour

The second thing school does not tell you: it points you exactly where everyone else is going.

Let us do some simple arithmetic.

Take a popular technology, open, accessible, carried by the trend of the moment. Say one hundred thousand trained candidates and ten thousand companies hiring for it.

The company gets a thousand applications for one post. It can afford to choose, so pay levels flatten. The candidate has to send many more applications, since every file is buried. The one who lands an interview is not necessarily the best: it is the one who had the most spare time to apply everywhere. The process drags on for months. Everyone concludes the market is saturated.

Now take a niche technology, unfashionable, proprietary, old. A thousand candidates, a hundred companies.

Those hundred companies do not have the thousand files. They have none. The two populations never meet. I published a job for an older web framework that is still much in demand and better paid than many fashionable roles. Zero local applications. The only replies came from South Asia, from professionals who had set up an automatic alert on that exact keyword.

That is the whole demonstration in one line. They have a filter. They are alerted whenever a post contains the term. They pick up

opportunities that are invisible to everyone else.

> **THE CALCULATION TO DO BEFORE CHOOSING**
>
> For any technology you are thinking of learning, measure the ratio between the number of openings and the number of profiles claiming it on your market. Any professional network lets you count this in ten minutes.
>
> A good ratio in a niche beats a terrible ratio in a prestigious specialism. Prestige does not pay the bills, scarcity does.

The standard objection is shame. I have heard it often: working on old technologies is supposedly demeaning, bad for your image, bad for your curriculum vitae.

Two answers.

The first is economic. The companies that own the client base, the cash flow and the critical processes were founded ten, twenty or thirty years ago. Their systems are not going to be replaced. They are going to be maintained. That need is structural and depends on no trend.

The second is strategic. When everyone differentiates in the same way, nobody differentiates. A profile that applies artificial intelligence to a legacy system stands out instantly in a pile of a hundred identical files. Not because it is better, because it is the only one.

The information trade

One last mechanism, and it needs naming plainly.

We live in a time when the person making money is often the one selling other people the information about how to make

money, while owning nothing at the start.

I have seen sets of instructions for a chat assistant sold for several hundred euros. Instructions produced by that same assistant. I have seen access to discussion groups sold at similar prices. I have seen improvised trainers charge serious money to pass on what they learned the week before.

None of this is illegal or unusual. It is the market.

The useful lesson is not moral, it is strategic. If packaging sells that well, then those who own the substance have an advantage they massively under use. You know how to build. All you are missing is the ability to package and to distribute.

> **ON THE GROUND**
>
> The most profitable skill I ever acquired is not technical. It is the ability to produce, in a few hours, a document that shows a director the diagnosis of his own system, with figures, a target architecture and concrete steps.
>
> That document is not sold. It is given away. That is exactly why it opens doors that ten sales messages would not open.

That is what is missing when you leave school. Not extra knowledge. A stance, some market arithmetic, and the ability to package what you already know how to do.

The next chapter is about what happens when you apply that arithmetic to a real decision, against everyone's advice.

◆

CHAPTER 3

Going against the crowd

> The entire first page of the rankings chose the same specialism. I took the one nobody wanted. What it cost me, and what it earned me.

End of third year. The moment when you choose your specialism.

The ranking sets the order of choice. I was well placed, I could take whatever I wanted.

The whole first page of the ranking took the same thing. Every single one. The most prestigious specialism, the one leading to advanced algorithms, research, international competitions, the logos everyone dreams about.

I took the least popular one. Information systems.

The reaction was immediate. I was told I was mad. That I was throwing away a rank I had worked hard for. That this specialism was for people who had no choice, not for those who did.

I answered something I regret in form, even if I stand by the substance. I suggested they were following the crowd without questioning it. That was clumsy and unfair: each of them had their reasons, and several have had very fine careers since. That sentence still cost me relationships, lastingly.

A stain on the first page of the rankings. That was exactly what I was after.

The reasoning behind the choice

It was not provocation. It was a reading of the market, made at twenty, with the information I had.

I asked myself three questions.

Where is the crowd heading? Towards algorithms, data science, artificial intelligence. All of the top of the ranking. Therefore all of my future competition, with exactly the same background, the same projects, the same keywords.

Where are the organisations that pay? In banks, insurance, logistics, public administration, large companies. They are not looking for researchers. They are looking for people who can understand an information system as a whole, connect a business need to a technical constraint, and hold together something that must not fall over.

Which specialism produces that profile? The one nobody wanted.

There was a third factor, subtler, that I understood later. Very narrow specialisms close doors at the same time as they open them. An over specialised profile is called in for a precise need. When that need does not exist, neither do they. A cross cutting profile, able to see the whole, is called on constantly, and above all moves towards decision making roles, because decisions require a broad view.

RULE

Deep specialisation is excellent for independent work, where you spread a rare skill across many clients. It is poor for employment, where you progress by widening your view, not by narrowing it.

Choose your specialism according to how you intend to work, not according to perceived prestige.

What the choice produced

The result, a few years later, is arithmetic.

Most of those on the first page who took the prestigious route followed the same path: further study abroad, then a doctorate. Not always by vocation, often by necessity. They came out with a narrow specialism and ran into an economy, here and elsewhere, that was not at the stage of applying what they could do.

Need cannot be forced. You can be scientifically right and have no market.

For me, the specialism unlocked the business world immediately. IT production in a bank, a technical lead role in outsourced services, then my own company. At every step, what counted was not depth in one field but the ability to hold the whole picture.

ON THE GROUND

I watched a brilliant student, twice top of his year, take a final year project inside a team that worked in incident mode from morning to night.

The subject was a standard one. The project could not be finished within the planned timetable.

Nobody was at fault. The team was competent and willing, but it had no continuous time to give to a long piece of work. In a stretched organisation, a student becomes one more request in an already full queue.

Talent does not make up for an environment that has no capacity to host it. Choose the supervisor and their real availability first, the subject second. And ask directly: how many hours a week will you be able to give me?

The general rule

From that decision I drew a rule I now apply almost everywhere.

Where the crowd is, I am not.

This is not a romantic pose. It is market arithmetic applied. When a hundred people compete for one resource, each person's value collapses. When nobody wants a resource that ten organisations need, the value of whoever holds it is set by them.

I applied that rule to my specialism. I then applied it to technologies, taking an interest in legacy systems while everyone rushed towards generative models. I applied it to markets, dropping one I knew perfectly for another where I knew nobody, because the first was full of friction and the second was open. I will apply it again.

Business is everywhere, including where there is nobody. Especially where there is nobody.

You have to be clear about the price.

Going where nobody goes means accepting being alone. Not only professionally: socially. You will not have a peer group going through the same difficulties. You will not have a model to follow. You will not have validation, not from classmates, not from institutions, not from professional networks.

I looked for a long time for a figure to follow in today's working world. Someone I could point to and say: there is the path, there is the method, I will draw on it. I did not find one. Not out of arrogance, from observation. I looked at whole cohorts, here and elsewhere, at careers, at legal statuses, at trajectories. I ended up accepting that there was no default path and that you have to draw your own.

That is uncomfortable at first. It is liberating afterwards, because the day you stop looking for validation, the direction of travel reverses. You are no longer chasing opportunities. They come, and you have the luxury of accepting or refusing them.

The ranking, again

A final note on rankings, since that is how this chapter began.

By choosing the least popular specialism, I finished first out of a hundred and two. The rank was easier to reach in a less crowded group. Some will see that as a trick. It is one, and I own it completely.

But there is better. I keep that title for life. It appears on my documents, it reassures a foreign contact in four seconds, it opens a conversation with a professor or a director. It costs nothing to maintain.

And meanwhile, the real value was built somewhere else: in the systems nobody wanted to touch.

> **RULE**
>
> Optimise the signals that cost nothing to maintain. Build the value where there is no competition. These are two different games and you have to play both.

The next chapter describes what happens when you walk into an engine room for the first time. Not a lecture, not a lab. The heart of a bank.

◆

CHAPTER 4

The engine room

> Cutting my teeth in the IT production team of a bank. What you understand when your hand is on the switch, and why it changes the way you see things for good.

My first steps as a professional were in the IT production team of a bank, part of a large international group.

The title is production engineer. What it means in practice: you are responsible for putting into production and keeping running all the applications and systems of the institution. Lending, foreign trade, customer onboarding, mobile banking, human resources, cheques and interbank clearing, international financial messaging.

You handle both the functional and technical administration of the whole estate. You qualify and certify every change, functional or technical, before it goes live. You take part in every project that touches the production application estate. You deal directly with national and international vendors during rollouts, version upgrades and third line incidents on third party systems.

You see everything. That is exactly why that stage counted for more than the five years of study before it.

You do not understand an information system by designing one. You understand it by keeping your hand on the switch while it runs.

What you find under the bonnet

What I found under the bonnet looked like no diagram from any course.

An old application core, written in technologies that appear on no current university syllabus. Batch processing everywhere. Files dropped into directories, picked up by schedulers, consolidated at regular intervals. Modern interfaces laid on top, giving the illusion of real time when underneath they were only triggering the same batch jobs as twenty years ago.

Nobody held the complete picture. Each team knew its own area. The overlaps were documented in the heads of two or three people whose departure would have been a major event.

And above all: it worked. Every day. For hundreds of thousands of people.

That is the first mental reversal I need to pass on. Coming out of school, you look at these systems with condescension. You tell yourself you would rebuild it all, cleanly, in a few months. You are wrong, and spectacularly so.

> **ON THE GROUND**
>
> That system carried decades of business rules, special cases, regulatory decisions, and fixes applied on a Friday evening to save a situation. None of those rules was written down anywhere except in the code.
>
> Rewriting it does not mean rewriting a program. It means rediscovering thousands of decisions nobody remembers, each of which, taken alone, looks absurd until you understand why it exists.

The pattern that keeps coming back

I will not tell any specific story. What I saw in that role is covered by professional confidentiality, and it will stay that way.

What I can describe is a generic pattern. It belongs to no institution in particular: it is documented across the whole of global banking security literature, it appears in the sector's audit frameworks, and I have since found it again in organisations of every trade and every country, including very well run ones.

The pattern is always the same. Someone opens an access to solve a real and urgent problem: letting a team work remotely, helping a partner, meeting a deadline. The opening is legitimate at the moment it is made. Then the person changes role, or leaves. The access stays.

Two years later, nobody knows it exists. It appears in no documentation. It is mentioned in no review. It works, silently.

What I took from that is worth more than any anecdote.

One. The most serious weaknesses are almost never sophisticated. They are doors left open for convenience, then forgotten. Real security is not cryptography, it is access review and organisational memory.

Two. Any organisation of a certain size, whatever its sector, accumulates these doors without knowing. The cause is never the competence of the teams, which is generally excellent and often better than its reputation. It is the compartmentalisation inherent to large structures, and the fact that a full access review fits into nobody's workload.

Three. Finding this kind of problem puts you in a delicate position. You are helping, and at the same time you are shining a light on something that belongs to a colleague's area. That is handled

through the formal channel, in writing, never naming anyone. This is not corridor politics, it is the only way to get the problem fixed rather than triggering a defence.

> **RULE**
>
> When you find a weakness in someone else's area, document it, raise it through the proper channel, and never try to gain reputation from it. The gain will be zero and the relational cost considerable.
>
> And never tell publicly what you saw inside. What you see from a position of trust stays there. It is a simple rule, and it has no exceptions.

The wear that comes with the job itself

There is a reality that is rarely discussed and explains a great deal.

In production teams, tiredness is structural. It comes from the nature of the job, not from the company that performs it.

That is neither a judgement on people nor a criticism of any management: it is an observation about a function. Working in incident mode wears you out, everywhere, in every company in the world that runs critical systems. You arrive in the morning not knowing what you will do, because the day will be set by whatever falls over. You never finish anything, you stabilise. You do not build, you repair.

After a few years that regime weighs on anyone. It is not a lack of motivation, it is the mechanical consequence of a permanent and unpredictable load.

The direct consequence, true across the whole profession: production absorbs everything, and innovation only happens by force. If someone does not carve out time deliberately, nothing happens. That observation belongs to no employer and no country: I have found it everywhere since, including at my own clients.

Forcing the issue

That is what I did, and it is probably the most useful part of that stage.

During my notice period I set up an operation with twelve engineering students, across six subjects, all connected to real problems in the organisation.

- A risk scoring engine applied to lending files.
- Automatic document recognition for foreign trade requests.
- Monitoring and automation of flows between servers.
- A model that suggests resolution paths from incident history.
- A central platform for capturing incident knowledge.
- Automated collection of information from several sources.

Nobody had asked for any of these. Nobody told me to go to the recruitment fair, select the candidates, define the subjects, supervise them, organise a closing ceremony. I did it because the space was empty.

Nobody told me to do it. That is exactly why it was possible.

Three observations came out of it.

One. A motivated, properly framed student produces in one month what others take a year to achieve. The limiting factor is almost never the student's ability. It is the supervisor's availability and the clarity of the brief.

Two. Good supervision, in this kind of environment, almost always rests on the personal willingness of an engineer giving up their own time, with nothing about it in their objectives or their pay. That is true everywhere, and it holds as long as that person holds.

Three. An operation like this has an unexpected side effect: it makes visible what is missing. Six innovation subjects means six functional gaps that any organisation, whichever it is, had been handling by hand because nobody had the time to get to them. Making that visible takes tact, and it must be done without ever putting anyone in a difficult position.

What that stage really gave me

I left grateful, with the feeling of having had access to something rare and of having been trained by serious people.

What I took away is not a skill in a tool. It is a mental map.

I know what a real information system looks like from the inside. I know that visible modernisation is most often a surface layer. I know that the organisations that keep the economy running are not equipped the way the articles suggest. I know that the real risk is not technical, it is human: the day the person who knows walks out.

And I know that this knowledge sells, because it is rare and because it reassures.

ON THE GROUND

When I started prospecting internationally, the thing that made the difference was never my command of a framework. It was the ability to describe, in three sentences, how the IT of a bank or a logistics firm actually works.

A director immediately recognises someone who has seen the engine room. You cannot fake that, and you cannot read it in a book.

Including this one. This book gives you the map. It does not replace the ground.

The next chapter covers the following step: two years working for foreign clients from Algiers, without ever changing time zone. That is where I understood that the market can come to us.

◆

CHAPTER 5

Outsourcing seen from the inside

> A few years delivering for foreign clients from Algiers. The founding discovery: the market can come to us. And the question that follows: where do you sit in the chain.

After the bank I joined an Algerian services company as technical lead. The next stage lasted a few years.

The model was this: a local team worked full time for a client based in France, on an outsourced basis. Other, more occasional assignments came from other foreign companies. Alongside that, we ran local projects on modern technologies.

That is where I saw the demonstration that started everything: the foreign markets came to us. Not the other way round. Nobody left. The work arrived, it was done here, it was delivered there.

That sentence sounds ordinary. It is the foundation of this whole book.

The foreign markets came to us, not the other way round. I only had to understand why, and how to reproduce that mechanism on my own account.

The job, in reality

What the role actually covered.

I ran the development and maintenance of legacy applications for the main client in France. I supervised occasional projects for others. I led the initiation, analysis, technical architecture and delivery of local projects on modern technologies: cloud platform, web framework, application interface, containers, continuous integration, artificial intelligence services.

I supervised final year projects and around twenty interns. I ran free public sessions, on cloud computing and operational practices on one side, on web development on the other. Several hundred participants in total, content built from the field, sessions recorded and left freely available.

I represented the company at recruitment fairs. I ran the technical interviews. I took part in business development meetings with French prospects, writing the technical proposals and the effort estimates.

That last point matters. That is where I learned to price, to defend an estimate, to understand how a foreign buyer thinks. No school teaches that.

The economics of nearshoring

Here is the mechanism, without decoration.

A European company has a development or maintenance need. It cannot find the skills locally, or it finds them at a cost it considers unbearable. It turns to a supplier in a country where labour costs are structurally lower, with a small time difference and a shared language.

The gap is large. The buyer pays a fraction of the local rate. The supplier is settled in foreign currency. The local engineer receives a local salary. In between sits the intermediary's margin, which pays

for the legal wrapper, the commercial relationship and the risk carried.

That is the classic economics of services companies, everywhere in the world. There is nothing illegitimate about it, and it renders a real service both to the buyer and to the engineer who would not reach that market alone. You simply need to know where you sit inside it.

> **THE GAP, IN ORDERS OF MAGNITUDE**
>
> An agency in a European capital charges a day of engineering at a rate most local structures cannot approach.
>
> A nearshore supplier charges a fraction of that, often between a third and a half, and remains very competitive.
>
> The engineer doing the work receives a share well below that fraction.
>
> None of this is scandalous or hidden: it is the model, and it is identical in Europe, in Asia and in the Americas. But when you are on the delivery side, you need to know exactly where you sit in that chain, and at what point it becomes rational to step out of it on your own account.

What I learned to do

Three decisive skills were built there.

Pricing. An estimate is not a calculation, it is a commitment. You learn to break work down, to identify what is genuinely unknown, to reserve time for what is never planned. Above all you learn to defend the number in front of someone who will try to reduce it.

Taking over what exists. Legacy applications from a French client, written fifteen or twenty years ago, with each client's business logic hard coded. You have no documentation, no original author and no room for error. That is where the skill I sell today was forged.

Talking to a buyer. Understanding that his problem is never the technical problem he states. His problem is a deadline, a risk, a budget, or a person who is about to leave. The technical part is only the vehicle.

> **ON THE GROUND**
>
> A client says he has a performance problem. You look. It is not performance. His long standing developer has left, nobody understands the module, so nobody touches it, and the debt builds up until the slowness becomes visible.
>
> The problem was not technical. It was human, and it almost always is.

The initiatives and their cost

I launched a great many things during those years. Public sessions, large scale supervision of interns, talks at engineering schools, coaching for project founders, a community project for an internship platform.

None of those initiatives was asked of me, and I want to be clear: nobody ever imposed anything on me. I carried them on my own initiative, on my own time, my evenings, my weekends, my holidays. It was my choice, and it came from ambition, not from pressure.

I have to be honest about the result.

In terms of learning, that is what shaped me most. In a few months, working with brilliant students from every specialism, I learned what I would not have learned in several years in a standard role.

In terms of immediate return, the picture is more mixed. The community project was not launched in the form planned. Initiatives given away quickly become an entitlement, which is the normal fate of anything free.

That is not a reproach to anyone. It is a mechanism, and it is better to know it before you start than to discover it afterwards.

And I ended up wearing myself out. The cause needs stating precisely: that load was self inflicted. It came from the pile of initiatives I had added to an already full role, without removing anything elsewhere and without telling anyone. My employer had nothing to do with it, never asked for any of it, and would very probably have told me to ease off had I said a word.

TRAP

Giving without a frame produces two effects, and you need to know both before you start.

The first is real and positive: you learn enormously, you build a reputation, you build relationships.

The second is mechanical: what you give becomes the expected standard. The day you stop, you do not go back to neutral, you become the one who stopped.

Give. But give knowing that you are setting a reference level you will not be able to come down from for free, and that the limit is yours to set, not anyone else's.

The central lesson

Those years gave me the conviction that shapes everything after.

Software work does not need physical closeness. It needs three things: real skill, a legal framework that reassures, and a communication channel.

We did the work from Algiers. The client was in France. He never came. We never went. It had worked for years.

So the question asked itself: if the work can be done here, if the client agrees to pay, if nobody needs to travel, why should the commercial relationship go through an intermediary?

If the work crosses the border, the invoice can cross it too.

The answer is one word: trust.

A foreign client does not only buy a skill. He buys a guarantee. He wants to know what happens if the supplier disappears, if the code contains a flaw, if the data leaks, if the project goes off the rails. He wants a counterparty that exists inside a legal framework he understands, with insurance he can verify, and compliance he can show to his own legal team.

That is exactly what the intermediary provides. It is not the skill, that is here. It is the framework.

The day I understood that, I understood the framework could be built. That it cost a few hundred euros and a few weeks. And that nothing required it to belong to somebody else.

That is the subject of part three of this book.

But before you build, you need to see the map. To understand how this market is really organised, who holds it, and why it is far more open than it looks.

◆

PART TWO

The map

You do not take a market you do not understand. This part describes the real structure of the sector: the pyramid that organises positions, the lock-in imposed by software vendors, the anatomy of the systems that keep whole countries running, how trends are manufactured, and what the large foreign markets are really worth when you look at them from Algiers.

CHAPTER 6

The pyramid

> Those who execute are at the bottom. Those who decide are in the middle. Those who own are at the top. And effort is not what moves you up.

There is a structure nobody draws for you and that you end up discovering alone, usually at the moment you start wondering why your effort is not producing the promised result.

I am going to draw it.

The three floors

At the bottom, those who execute. They do the work. They keep the systems running, resolve incidents, deliver features, absorb the pressure. They create the value, and their pay is the most constrained, because there are many of them and they are interchangeable from the buyer's point of view.

In the middle, those who decide. They arbitrate, prioritise, allocate. Their value does not come from what they produce but from their position in the flow of information. They see several teams, several projects, several stakes. That cross cutting view is exactly what the executor does not have.

At the top, those who own. They hold the structure that invoices. They capture the gap between the selling price and the cost of production. They are not necessarily more competent. They took a risk at some point, and they occupy the position that captures.

Effort moves you up inside a floor. It does not move you between floors.

That is the most important sentence in this chapter.

The story we were told throughout school is that effort is rewarded and hard work takes you up. That is true inside a floor. An executor who works hard becomes a better executor, better paid, better regarded. He remains an executor.

Changing floor is not produced by effort. It is produced by a change of position.

The paradox of effort

Looking at a great many careers, I noticed an uncomfortable correlation: financial success is often inversely proportional to the intensity of the effort put in.

Look at who actually makes money today.

People sell sets of instructions for a chat assistant at several hundred euros, instructions generated by the assistant itself. People sell access to private discussion groups at similar prices. People sell training on how to make money, having built nothing else than that training.

This is not fraud. These people have understood something engineers refuse to accept: perceived value is set by distribution, not by cost of production.

> **RULE**
>
> The market does not pay for difficulty. It pays for scarcity and access.
>
> Difficult but common work pays badly. Easy but rare work pays well. Difficult and rare work pays very well, and it is the only box an engineer should agree to stand in.

The question that changes everything

At some point I stopped asking how to work better. I started asking something else.

Who deserves this effort?

I looked seriously. I looked at whole cohorts, here and abroad. I analysed careers, legal statuses, positions. I used paid professional research tools to map entire populations of engineers and see what became of them. I was looking for a figure to follow.

I did not find one.

That is not arrogance, it is an observation that took me time to accept. Today's working world does not produce many role models. It produces positions.

So the next question becomes unavoidable: who are you doing those sleepless nights for? For which institution are you giving up your weekends? Whose validation are you waiting for?

If the answer is a system you do not own and that will not reward you beyond a certain ceiling, then the effort is misplaced. Not useless: misplaced.

When you stop looking for validation, nature abhorring a vacuum, you become the one who validates.

How intermediaries work

A word on platforms, because they crystallise the mechanism.

On a large freelancing platform, you pay for the right to apply. You buy tokens to bid for work, with no guarantee of being read.

On an online teaching platform, you hand over a large share of your revenue, and you must give your first trial lessons free, even though the client has paid the platform for them. You end up working unpaid for a significant part of your time.

I registered on several of these platforms on the employer side, to see. The view from that side is very instructive: you get access to everything, you measure the asymmetry, you understand the model.

These are not scams. They are capture positions, methodically built. Whoever set them up understood before everyone else that value does not sit in production but in controlling access.

ON THE GROUND

Look at what a company director receives in his professional inbox. Every day, a dozen approaches from suppliers offering developers, agencies, services.

Now look at what an engineer looking for work receives. Nothing.

Both populations exist. They never meet. The whole value of the intermediary lies in holding the meeting point.

You can hold that point yourself. That is the subject of part four of this book.

Getting out of the pyramid

There are three ways to change floor. They are not equivalent.

Through internal promotion. You move up the hierarchy of an organisation. That is the classic route, it is slow, it requires political intelligence, and it works. Watch one counter intuitive point: the more specialised you are, the harder it is, because a decision making post requires a broad view. Deep specialisation slows hierarchical progress.

Through scarcity. You become the only one, or one of the few, able to do something several organisations need. Your bargaining power changes in nature. You no longer ask, you are approached.

Through ownership. You create the structure that invoices. You no longer capture a salary, you capture a margin. It is the only route that genuinely changes the floor, and the only one where you carry the risk.

> **RULE**
>
> A productivity gain you create and do not invoice is captured by someone else. Always.
>
> If you divide by thirty the time a task takes inside an organisation that pays you by time, you have just given away a margin. If you do the same in your own structure, you have just multiplied yours.
>
> The technical act is identical. The economic result is opposite.

The honest calculation

I am not saying you should leave employment. That would be stupid and dangerous advice.

Employment buys stability, access to systems you would never otherwise see, and accelerated training nobody else will give you. My year in bank production is worth several years of learning on my own. I would not trade it for anything.

What I am saying is more precise.

You need to know which floor you are on. You need to know the maximum that floor can give you, and when you reach that ceiling. You need to know that any extra effort beyond the ceiling is not captured by you.

And you need to prepare the change of floor while you are still protected, not afterwards.

That is exactly what I did. I opened a self employed status while still in post. I set up the foreign company while still invoicing. I never jumped without looking at where I would land.

Do not leave a floor. Build the next one, then move.

The next chapter describes who occupies the top of that pyramid in our sector, and how they stay there. This is not a conspiracy theory. It is a well documented industrial strategy, and it is worth understanding in detail.

◆

CHAPTER 7

The lock-in

> How a software vendor takes control of a whole sector, buys its competitors in order to shut them down, and turns a piece of software into a permanent toll.

The mechanism needs naming, without naming the companies.

In several critical sectors, banking, insurance, human resources, logistics, public administration, the core software is not built by the organisation that uses it. It is bought from a vendor. That vendor is often the same one for a large part of the sector, across a whole country, sometimes several.

There is nothing abnormal about that. Nobody builds their own operating system. But when that software becomes the beating heart of a regulated sector, the relationship changes in nature.

The mechanism in four layers

Layer one: technical dependency.

The software holds the data. All of it. Customers, accounts, movements, regulatory history. Every other application in the organisation is only an interface plugged into it.

Replacing that core is not an IT project, it is open heart surgery on an organisation that must keep operating throughout. Those projects are long, expensive, and their failure rate is well known. Everyone knows it, including those considering it.

So the dependency is not contractual. It is physical.

Layer two: the regulatory barrier.

In a regulated sector, the software must comply with a stack of standards, certifications and international exchange protocols. Obtaining those takes years and considerable resources.

The practical consequence, and I have seen it myself: when a new organisation is created in these sectors, the question asked by the authority is not what is your architecture. It is which system are you using. And the expected answer is on a very short list.

There is no fourth option. And the mere fact that everyone knows it is enough to stop anyone trying.

That is the most effective trap, because it needs no enforcement to work. It works by anticipation. A competent team could technically rebuild a modern core in a few months with today's tools. It does not, because it knows in advance it will not get through approval.

Layer three: buying to shut down.

This is the least known and most decisive part.

When a smaller competitor starts to look like a threat, with a more modern architecture, lower costs, a more credible story for buyers, the large vendor does not fight it in the market. It buys it.

Then it switches it off. It puts the acquired product into maintenance only, announces an end of life, and migrates the customer base onto its own offer.

The result is mechanical and visible over time:

- The competitor disappears.
- Its customers are absorbed, often forced to migrate.

- The price war that was starting stops.
- The market returns to its previous order, with one player fewer.

This is neither illegal, nor hidden, nor unusual. It is a consolidation strategy, it is visible in financial statements, it has a name in business schools. It is simply called buying market stability.

> **WHAT TO TAKE FROM THIS**
>
> In a locked market, innovation is not rewarded with market share. It is rewarded with an acquisition offer.
>
> That completely changes the strategy for a new entrant. If your plan is to compete head on with the dominant player, your best outcome is to be bought and switched off.
>
> The only workable position is the one it does not want to occupy.

Layer four: cosmetic modernisation.

The large vendor has to sell something new on top of an old base. So it adds visible layers: modern portals, mobile applications, application gateways, and more recently artificial intelligence features.

Underneath, the engine does not change. Batch processing stays batch processing. Dropped files stay dropped files.

Adding artificial intelligence on the surface serves two purposes. It allows an innovative solution to be sold. And it adds a further layer of complexity that only the vendor holds the key to, giving one more reason to call them back and pay them.

ON THE GROUND

An instant transfer between two institutions, in an old system, does not mean the core has been rebuilt.

It means a layer has been placed in front of the core that authorises the operation, calculates a difference and displays the result immediately. The actual settlement still goes through the end of day batch consolidation.

The user sees real time. The system does what it always did. All the modernisation sits in a few thousand lines placed in front of an unchanged building.

And it is good architecture. This is not cheating, it is pragmatism. The problem is not the method, it is the price it is charged at when only one player can carry it out.

What this means for a country

When a country puts the core of its critical sectors in the hands of a single foreign vendor, it creates a single point of failure.

The risk is not theoretical. A country in the region saw its financial institutions stuck for years on an obsolete version, with no updates and no support, for geopolitical reasons entirely outside the technology. The software worked. Nobody could make it evolve any more.

The other consequence is monetary, and it is large. These licences, services and days of work are paid in foreign currency. In a country where currency is a constrained resource, every euro that leaves for a foreign vendor is a euro that funds nothing else.

Add another layer to that reality. It often happens that the European vendor subcontracts the actual work to teams based in other countries. When a local organisation opens a third line ticket, the person at the other end is not necessarily in the vendor's home country.

The full chain looks like this: an organisation pays the vendor at home country rates, and the vendor subcontracts elsewhere at a much lower cost. The intermediary does add real value, but it is legal and commercial: the contractual wrapper, the guarantee, the client relationship. What is worth noticing is that the engineer producing the technical value receives only a fraction of it, and that no economic law requires that fraction to stay small.

It is not a competence problem. It is a position in the chain problem.

What it opens up

Here is the useful part, and it is counter intuitive.

This lock-in creates the most solid opportunity I know.

The dominant vendor occupies the core. It does not want to occupy the edges, they are not profitable at its scale. And the edges are vast:

- The satellite applications around the core, often built in house fifteen years ago by someone who has left.
- Organisations too small to interest the vendor but with exactly the same problems.

- Systems that were bought and then put into end of life, whose users are abandoned and worried.
- Proprietary technologies that the vendor itself struggles to keep staffed.

That is where I placed myself. Not in head on competition. In the blind spots.

> **RULE**
>
> In a locked market, do not try to take the centre. Take what the centre does not want.
>
> The centre is defended by billions. The blind spots are defended by nobody, and they are in pain.

A practical note to finish.

These players watch their environment. When I created my company and published an explicit position on legacy modernisation, several views of my professional profile came within days from people working at one of these large vendors.

I did not know them. Nobody introduced me. They have monitoring tools on the keywords that touch their model, and those tools did their job.

That is not threatening. It is even flattering. But it teaches you something useful: your positioning is a signal, and signals are heard by people you did not invite.

Know what you are broadcasting.

◆

CHAPTER 8

Anatomy of a system that runs a country

> What is really under the bonnet: the languages, the formats, the batch processing, and why full rewrites almost always fail.

This chapter is technical. If you are a non technical director, read it anyway: it will explain why your team looks like it is doing nothing while working very hard.

The families

Four broad families make up most of what I call critical legacy.

The language of large business systems. Born in the nineteen sixties, designed to handle business records at volume. It still runs today in banking cores, payroll systems, insurance and the tax administration of several countries.

Its defining features:

- Data description blocks are shared between programs. A change in one place spreads everywhere.
- Memory areas can be redefined, meaning the same area is read in several different ways depending on context. That is a nightmare for any automatic analysis tool.
- Numbers are stored in a packed format that bears no relation to a modern representation.

- Database queries are written directly in the code, with no abstraction layer.
- Files are sequential or indexed in a proprietary format.
- Orchestration goes through a dedicated job control language.

Common order of magnitude: from a few hundred thousand to several million lines, with no documentation.

Proprietary mid range integrated systems. Very common in insurance, logistics and distribution. They use a language whose distinctive feature is an implicit cycle: the program does not describe its main loop, the system triggers processing as it reads records.

That period elegance becomes a major obstacle. Reproducing that behaviour in a modern language means making it explicit, so understanding it perfectly, so reconstructing logic the code does not state.

Proprietary rapid development environments. Extremely widespread among mid sized companies, particularly in France. In the nineteen nineties and two thousands they made it possible to build complete business applications very quickly, with a graphical interface and built in data access.

The problem today is twofold. The sources are in a proprietary format that standard tools cannot read. Version control is proprietary and incompatible with modern practice.

The consequence is decisive and few people measure it: no mainstream development assistant can work natively on that code. The whole market for intelligent assistance stops at the door.

Older web applications. A very widespread scripting language, in a version that has not been supported for years, with no framework, tangled code, queries built by concatenation and publicly documented vulnerabilities. Or an elegant framework from the late

two thousands, frozen in a version whose entire ecosystem has moved on.

> **WHAT THEY HAVE IN COMMON**
>
> These four families share exactly the same symptoms:
>
> - No usable documentation.
> - No original author still present.
> - No automated tests.
> - Business logic scattered through the code, never formalised anywhere else.
> - One or two people in the organisation able to touch it, and they are close to retirement or to leaving.
>
> The risk is not that the system falls over. It runs very well. The risk is that the last person who understands it walks away.

The pattern that kills

There is one design pattern I find in almost every system I open, in every sector.

Hard coded customer specialisation.

In practice, inside the business logic, you find explicit tests on the customer identity: if the customer is this one, apply this particular rule. Then another customer asks for a variant. Then a third.

After fifteen years and a hundred customers, the central function of the application is a stack of conditional branches encoding the commercial history of the company.

Why does this happen? Not through incompetence. Through commercial pressure. Every customer request was an invoice.

Every invoice had a deadline. The quick fix was always to add a branch.

> **The code faithfully tells the story of commercial compromises. It is even the only place where that story is written down.**

The consequences are severe.

Testing becomes impossible. Shipping a change for one customer breaks another's flow. Since there are no automated tests, you find out in production.

Versions multiply. Some customers stay on an old version because upgrading would break their specifics. The team maintains several branches in parallel.

Recruitment becomes impossible. You need someone willing to work on that technology, and willing on top of that to enter this accidental complexity.

Why the full rewrite fails

At this point the temptation is irresistible: start again from scratch, cleanly, with modern technology.

That is almost always a mistake, and here is why.

One. You are not rewriting a program, you are rediscovering thousands of undocumented decisions. Every conditional branch corresponds to a real requirement. You do not know which. The only way to find out is to remove it and wait for the complaint.

Two. During the rewrite, the old system must keep running and keep evolving. So you maintain two systems, with the same team, for several years. The target chases a moving source.

Three. Automatic translation produces code that is technically modern and structurally unreadable. You get the old language written with the new syntax. The result is worse than the original, because it has lost the conventions the old hands knew without having gained a modern structure.

Four. The budget is voted on an initial estimate that does not survive contact with reality. When the overrun becomes visible, the project is stopped halfway. The organisation is left with two incomplete systems.

> **TRAP**
>
> The grand story of the rewrite is psychologically comfortable. Saying we are going to rebuild everything relieves an exhausted team, even when it is the worst possible decision.
>
> Technical debt is as much a psychological subject as a technical one. The job of good advice is often to remove that relief and propose something less heroic and far safer.

What does work

The method that works is gradual and unspectacular.

Step zero. Understand before touching. Map the modules, extract the business rules, identify the customer branches, spot the dead code. This is archaeology, and it is where intelligent assistance brings the most.

Step one. Stabilise. Before any modernisation, stop the bleeding. Put monitoring in place, review access, patch what is exposed, document the operating procedures. The client must be able to sleep first.

Step two. Put a facade in front. An application interface in front of the existing system. New needs go through the facade. The old system keeps running behind it, unchanged.

Step three. Strangle gradually. Function by function, the logic behind the facade moves to a modern module. The facade decides: this function has moved to the new system, that one still goes to the old. Nobody outside sees the difference.

Step four. Decouple the data. Often the database becomes the meeting point between the two worlds. The new module reads and writes where the old one reads and writes. That is not academically pure, it is operationally very effective, and it avoids a high risk data migration.

Step five. Extract the rules. The per customer conditional branches move into a configuration mechanism. From then on, adding a customer no longer requires changing code.

> **ON THE GROUND**
>
> What gives a director confidence is not the promise of a final result. It is the demonstration that at every stage you can stop without damage.
>
> A well designed gradual modernisation is reversible at any point. A full rewrite never is, and that is exactly what makes it terrifying to fund.
>
> Sell reversibility. That is what people are buying.

The software as a service story

One final point, because it says a lot about what modernisation really is.

Many vendors announce that their product is moving to an online service. Look under the bonnet.

In most cases it is not a shared architecture between customers. It is a complete, isolated instance deployed automatically for each customer, with a gateway in front for routing and authentication.

The software has not changed. The deployment has been industrialised.

Is that wrong? No. It is often the rational choice: a genuinely shared architecture is prohibitively expensive for a mid sized vendor, and per customer isolation offers real guarantees.

But you need to be able to read it. When a buyer believes he is acquiring a modern platform while acquiring an automated deployment of old software, the gap between the price paid and the value received is considerable.

Being able to read an architecture behind a sales pitch is, in itself, a billable skill.

That is in fact what I most often sell first: not development, but a reading. And an honest reading opens more doors than ten prospecting messages.

The next chapter explains why almost nobody stands on this ground, and how a whole generation of engineers was pointed at exactly the same place.

CHAPTER 9

Trends as a product

> Two hundred and fifty thousand people enrolled on a single course. A whole generation pointed at the same place. And a market that, meanwhile, is looking for something else.

A professional contact once asked my opinion on a paid online course about a recent technology.

I looked at the number of people enrolled. Around two hundred and fifty thousand.

Two hundred and fifty thousand people on one course, among dozens of similar courses, on one platform among many, in an ecosystem that also includes university programmes, bootcamps and vendor certifications.

That figure is the starting point of any thinking about the technical job market.

A skill taught to two hundred and fifty thousand people is no longer a skill. It is a baseline requirement.

The training industry

Let us look honestly at the economics.

A technology becomes visible. It is open, accessible, well documented, backed by prestigious companies. Demand for learning explodes.

An industry forms around it immediately. Course platforms, certifications, improvised trainers, paid content, monetised communities. That industry has a structural interest: that as many people as possible learn that precise technology.

The training industry does not sell jobs. It sells the hope of a job. Those are two completely different products, and only the second is guaranteed on delivery.

Result: hundreds of thousands of people acquire the same skill, at the same time, on the same market.

> **THE ARITHMETIC, ONCE AGAIN**
>
> Technology in high demand among candidates: a hundred thousand trained profiles, ten thousand companies hiring.
>
> The company receives a thousand files for one post. It chooses. Pay flattens mechanically.
>
> The candidate has to multiply applications. The process takes months. Everyone concludes the market is saturated.
>
> Niche technology: a thousand profiles, a hundred companies.
>
> The companies have no files at all. The two populations never meet. The candidate who sets up an alert on the rare keyword picks up everything.

The experiment that convinced me

I ran the test myself, twice, a few weeks apart.

I posted a job for an older but still used web framework, properly paid. Result on the local market: zero applications from anyone who genuinely knew the technology. One person with some notions. Two applications from South Asia, including one experienced expert.

I posted a job for a popular technology stack. Result: more than fifty applications within days, with very strong profiles, some more experienced than me. I had to turn down the vast majority, reluctantly.

Same author, same channel, same audience, a few weeks apart.

The only variable was the keyword.

> **THE QUESTION TO ASK**
>
> Why did a professional thousands of kilometres away apply to a post published on a local feed, in a language that is not his own?
>
> Because he had a filter. An alert set on a rare keyword, permanent monitoring of opportunities invisible to everyone else.
>
> That is not talent. It is method, and it takes ten minutes to set up.

Shame, the invisible brake

The real obstacle is not about information. It is psychological.

I have offered work on old technologies, better paid than equivalent roles on fashionable ones. I have seen candidates hesitate, then refuse, always with the same argument: it would look bad on their profile.

That shame is manufactured. It is the direct product of the training industry and of the constant staging of careers on social

networks.

It costs those who suffer from it a great deal.

I saw it in its clearest form. Someone who had gone abroad, with a strong profile, was actively looking for part time work in technology. They could not find any. I was considering offering them a proper contract.

It did not happen. The people around them felt that working on legacy systems would be a step backwards in their career.

They preferred to wait for something else rather than accept something poorly regarded. That is a trade off many people make, and it is understandable.

Conformity costs more than failure. Failure at least teaches you something.

What the market actually wants

Here, from what I see in my own field, are the profiles organisations struggle most to find.

Data engineer in the infrastructure sense. Not data science. The person who administers the databases, the backups, the restores, the replication, the loading pipelines. There is almost no dedicated course. Organisations fight over them.

Systems and network engineer. A permanent need, never covered. A demanding skill, socially undervalued, needed everywhere.

Project manager and information systems analyst. So rare as to be almost impossible to find. I have been asked to find one and had

nobody to put forward in my entire network.

Genuinely full stack developer. Not front end, not back end: able to take a project from need to deployment. That is what lets you hold a durable post rather than string together fragmented assignments.

Legacy technology specialist. The subject of this book. Almost no candidates, a structural need.

Conversely, the specialisms where the supply of profiles far exceeds local demand are exactly the ones filling the lecture halls.

> RULE
>
> Never choose a specialism on perceived prestige or media presence. Choose it by calculating the ratio between the number of openings and the number of profiles on your real market.
>
> Then check how stable the need is over time. A rare skill tied to a passing trend is a delayed trap.

The order of priorities

There is one more layer, and it applies to the whole world, not to any one country.

In many organisations, the communication about artificial intelligence moves faster than its actual deployment. A chat interface on a website, a promising pilot, a component bought from a vendor: that is a start, it is not yet a transformation.

Meanwhile the technical team spends most of its energy keeping the existing estate running. That is difficult work, often invisible, and everything else depends on it.

So I ask a plain question to anyone choosing a final year project in a company: what is the real place of this subject in the priorities of the organisation hosting you?

If the answer is marginal, the work may be brilliant and perfectly defended, but it will not be picked up after you leave. That is nobody's fault: it is a question of sequence.

Solid foundations first. Artificial intelligence afterwards, and it will impose itself on its own.

The turnaround

I will finish with the positive point, because it is considerable.

This massive concentration of the crowd on the same skills creates the most accessible opportunity I have come across.

A recruiter opening a hundred files for a technical post reads the same thing a hundred times: the same libraries, the same demo projects, the same certifications, the same connectors to the same services.

The file that applies artificial intelligence to a legacy system stands out immediately. Not because it is technically better. Because it is the only one.

The same is true for a company. A business positioned on the maintenance and modernisation of old systems ranks, in professional directories, ahead of thousands of businesses positioned only on the technologies of the moment. Not through merit: through absence of competition on the term.

ON THE GROUND

The same holds for academic research. A student who writes to a laboratory director mentioning the application of modern methods to legacy systems stands out immediately from thirty identical applications.

The vast majority of teachers and decision makers over forty are deeply tired of the fashionable discourse. Talking to them about something else is an immediate and free advantage.

The next chapter leaves the national frame and looks at the large foreign markets as they really are, from Algiers, without illusions.

◆

CHAPTER 10

The world seen from Algiers

> France, the United Kingdom, India and Pakistan, Eastern Europe, the Americas. What these markets are worth when you look at them without illusions, with the numbers and the friction.

I have prospected, talked, negotiated, refused and delivered across several markets. Here is what I took from them, without diplomacy.

The French market

This is the one I know best. Two years working for French buyers on an outsourced basis, then several months of direct prospecting.

What works. A shared language, no time difference, real cultural closeness, a dense fabric of mid sized companies running old proprietary systems. The technical need exists on a large scale.

What blocks. A significant share of the people I dealt with put form before substance.

I have seen the demonstration several times. I carry out a preliminary technical assessment from public information, I spot an obsolescence or a weakness, I offer to fix it.

On the British market, the conversation goes immediately to risk, speed of execution and the terms of a trial.

On the French market, I was frequently told about internal procedures, and the transparency of my approach was received as a threat rather than as a service.

I do not sell desk time or administrative presence. I sell the resolution of technical debt.

There is a local vocabulary for this: savoir-être, roughly meaning how you conduct yourself. It is an elegant word that sometimes covers very concrete things. Not turning your camera on in a remote meeting can be held against you as poor conduct, even when the output is faultless.

I turned down several opportunities on that market. Not for lack of time. Because of a mismatch of values: we put operational excellence first, the other side put appearance first.

One necessary clarification. I am not saying that market is bad. I am saying it is poorly aligned with the way I work. Others do very well there, precisely because they can play that register. Know yourself before you choose your ground.

ON THE GROUND

Through supporting a French services company, I had access to its full management accounts.

The finding was brutal. Value added tax is charged on invoices issued, whether or not the client has paid. Looking at the history over several years, more than half the clients had never settled.

A company that works, invoices, pays its staff out of its own cash, and whose clients vanish. In business to business trade.

Remember this: the main risk on a foreign market is not failing to find a client. It is finding a client who does not pay.

The British market

I moved my business towards this market after several months of exploration. Here is why, and here is what nobody tells you.

The accessibility is astonishing. That is what surprised me most. You write to a director, he replies. You write to a director of a large group, he replies. No intermediary, no introduction, no status games.

I have spoken with a former regional director of a public service covering over a million people who had become a coach, with a commercial director used to growing multi million pound businesses, with founders of independent firms, with heads of recognised organisations. No positional ego. You bring something, the conversation opens.

Silence is cultural. You publish, nobody reacts publicly. Zero comments, two or three signals of approval. You conclude that your content interests nobody.

That is wrong. They read, they do not react, and they write to you privately. Never judge your British traction by visible engagement.

Decisions are slow, then execution is fast. People do not commit verbally until they have decided. After that it moves quickly.

Proof beats talk. What kills an opportunity: vague promises, generic approaches, no demonstration. What opens one: a concrete diagnosis, total transparency about risk, a verifiable legal framework.

WHAT NOBODY TELLS YOU ABOUT THIS MARKET

Widespread professional exhaustion. People are worn out, customer service is poor, administration is slow, layered subcontracting is everywhere. There is even a local expression for this general decline in services.

I watched a director, a former local authority manager with hundreds of clients to her name, end up in housing difficulty and cancel our conversations because of it. The system she had served had not protected her.

The eldorado myth does not survive three months of direct observation. It is an excellent market to sell into. It is a much less obvious market to live in.

The position of foreign students. Very high tuition fees, no right to set up a company, no right to work as a freelancer, a strict cap on paid hours. Many end up in unskilled work to balance their budget.

The paradox is striking: from Algiers I can create a British company, open business accounts and invoice legally, while a student physically present there is not allowed to.

It is better to plug into that market than to move there.

India and Pakistan

One observation worth stating plainly: in the technical functions of many British organisations, a very large share of the teams comes from India and Pakistan, whether on site or through subcontracting.

Two lessons.

One. These are often the teams doing the most work in the chain, and they hold a solid structural position. They have captured support, production and maintenance, which is to say exactly the functions nobody wants and everybody depends on.

Two. That success is no accident. It rests on systematic monitoring, willingness to take unglamorous assignments, collective organisation and the capacity to absorb large volumes.

There is a model to study there rather than to comment on. We have exactly the same strengths and we do not use them.

> **WHAT I RECEIVE EVERY DAY**
>
> Since setting up my company I receive around a dozen commercial approaches a day from foreign suppliers, mostly Indian and Pakistani: development agencies, services companies, licence resellers.
>
> The messages are alike: company profile, list of skills, rates, promise of immediate availability.
>
> Many are poor quality. Some rely on expensive sending address rotation services.
>
> And yet they persist, which means it works. Volume makes up for quality. That is a market lesson, even if it is not the model I chose.

Eastern Europe and the Gulf

Eastern Europe is very close to the British market in pragmatism. Exchanges are direct, technical, without detours. I have been approached by people who built tools used by millions and who simply wanted to talk through a problem.

The Gulf has a particular profile: real resources, a need for expertise, a different relationship with time, and strong attention to the personal relationship before the transaction. It is a market where a recommendation counts more than a file.

The Americas

I explored one route with a partner based in a Caribbean territory attached to the United States, holding American citizenship and working in two languages. The interest was direct: testing the North American market through a local anchor point, without immediately committing the main structure.

That arrangement is entirely legal and it is instructive: entering a market does not require setting up there. It requires a credible relay.

A broader remark on the American players I met. I was approached to sell old software assets for several thousand dollars. I was approached to sell my company while it was barely starting, as part of industrialised campaigns buying up companies that already had a complete legal and banking framework.

What sells most easily internationally is not skill. It is the framework.

The comparison table

Market	Main strength	Main friction	What I do with it
France	Language, closeness, real need	Form before substance, late payment	Selective, never dependent
United Kingdom	Access, pragmatism, legal framework	Exhaustion, slow administration	Main market
India and Pakistan	Volume, organisation, monitoring	Price competition	A model to study
Eastern Europe	Pragmatism, directness	Shallower market	Occasional partnerships
Gulf	Resources, need for expertise	Different sense of time	Gradual exploration
Americas	Size, speed of decision	Distance, heavy legal framework	Through a local anchor

The conclusion that counts

Having looked at all this fairly closely, I am going to say something surprising.

I am disappointed by the international scene.

We are better here. Not in resources, not in organisation, not in marketing. In raw ability, in resourcefulness, in resistance to hardship and in depth of training.

What I saw abroad was well oiled systems staffed by people who are often tired, often conformist, sometimes technically disappointing relative to their stated position.

> **RULE**
>
> Do not go abroad looking for a level you already have. Go looking for purchasing power you do not have.
>
> Those are two completely different approaches. The first requires moving. The second requires a legal structure and a sales channel.

That is the map. You now know how the sector is organised, who holds it, what is under the bonnet, why everyone is looking elsewhere, and what the large markets are worth.

The next part is the toolkit. How you build, in practice, from home, the structure that lets you go after all of this.

◆

PART THREE

The toolkit

This is where it becomes practical. Setting up a company abroad without going there, building the legal armour that reassures a buyer, getting paid in spite of banking friction, obtaining industrial grade infrastructure for almost nothing, and turning artificial intelligence into a team. Five chapters, real costs, concrete steps.

CHAPTER 11

Setting up abroad without going there

> A few hundred euros, a few weeks, no travel. What the structure actually buys you, and what it does not.

Let us start by taking apart a belief.

Setting up a company in a foreign country is neither difficult, nor expensive, nor reserved for a particular category of people. It is an administrative formality you can complete online, in a few days, for an amount that fits in a personal budget.

The price of the formality is trivial compared with what it unlocks. That is not the point. The point is everything that comes after.

A company is a legal vehicle. You can change the bodywork whenever you like.

What the structure actually buys

It does not buy skill. You already have that.

It does not buy clients. They do not arrive because you exist.

It buys four precise things, and they are exactly the four the intermediary was charging you for.

One. A legal framework the buyer can read. When a director hands you a critical system, he is not asking whether you can code. He is asking what happens if things go wrong. A contract with a

private individual in a country he does not know gives him no usable recourse. A contract with a company registered in his own jurisdiction does.

Two. Verifiable existence. The company register is public. He can read the number, the date of incorporation, the address, the director. That check takes thirty seconds and it changes everything.

Three. Full contracting capacity. A company signs confidentiality agreements, service contracts, data processing agreements. It can take out professional insurance. It can open business accounts. It can hire in the country.

Four. Access to programmes. A large share of the schemes for young companies require a legal entity in an eligible jurisdiction. Those schemes hand out infrastructure credits and tools whose combined value far exceeds the cost of incorporation. Chapter 14 is devoted to that.

Whether the arrangement is legal

An essential point, because it is the one that worries people most.

Being tax resident in one country and director of a company registered in another is perfectly normal. Millions of people worldwide are in that situation.

What is regulated is the financial flows. Capital must have a legitimate origin. A company can be set up with a symbolic capital, released by the first client payment. The resident continues to declare income and pay contributions in their country of residence, under whichever status applies to them.

> **RULE**
>
> The arrangement must be simple, documented and defensible in front of any authority, on both sides.
>
> If you need more than three sentences to explain your structure, it is too complicated. If you need to hide part of how it works, it is a bad one. Simplify until it is boring.

A word on perception. This kind of arrangement looks, from a distance, like far less respectable schemes. You need to be aware of that and neutralise it with transparency: a real and public address, real activity, real invoicing, declared compliance, insurance in place.

This is not a shell company. It is a services company operating from another country, which is entirely ordinary.

The concrete building blocks

Here is what I actually put in place, in order.

Incorporation. Done online through a formations agent. The simplest available form, symbolic capital, one director. A few days.

Registered office. A professional address in the country, with mail forwarding. Modest annual cost. This is the address that appears on the public register.

Watch one point of fragility, and I identified it early: that address depends on a supplier. They can end the relationship. Plan a fallback before you need it.

Digital identity. Domain name, website, professional email addresses. This is the control point that makes everything else credible. A dedicated domain costs a few dozen euros a year.

Business accounts. I devote a whole chapter to this, because by far it is the hardest part. Immediate rule: open several, from the start, before you need them.

Insurance and compliance. Next chapter.

> **ORDER OF MAGNITUDE FOR STARTING UP**
>
> Incorporation and formalities: a one off cost, at most a few hundred euros depending on the agent.
>
> Registered office: billed annually, a few hundred euros.
>
> Professional indemnity insurance: a few dozen euros a month for industrial grade cover.
>
> Registration with the data protection authority: a few dozen euros a year.
>
> Domain name and professional email: a few dozen euros a year.
>
> Total for the first year: an amount that fits within a personal budget, with no loan, no investor, no partner.
>
> That is the most important point in this chapter. The barrier to entry is not financial. It is mental.

What the structure does not buy

We should be just as clear about the other side.

It does not buy automatic credibility. A registration number convinces nobody on its own. Credibility is built through proof: deliverables, recommendations, expertise demonstrated in public. The structure makes that proof enforceable, it does not replace it.

It does not buy a network. You start from zero. The structure lets you enter certain rooms, it introduces you to nobody once you

are inside.

It does not buy cash. On the contrary, it creates fixed costs. They have to be covered from month one.

It does not protect you from yourself. A bad commitment remains a bad commitment.

> **ON THE GROUND**
>
> The first genuinely effective signal was not the incorporation. It was the moment I started publishing precise technical analyses on subjects nobody was covering.
>
> Incorporation made those analyses sellable. It did not produce them.
>
> The correct order is: skill, then public proof, then structure. Many people do the reverse and end up with an empty shell.

The unexpected ranking

A useful story about what a clear position produces.

A few months after incorporation, my company appeared in the rankings of an international platform listing more than a million technology companies. Two months in a row, in two different categories, at positions that are very visible at country and world level.

Those rankings are read by investors. They place the company ahead of heavily funded businesses.

I have raised no money, I am not looking for any, and I do not consider myself a startup. So the question is fair: how does a company with no funding and no track record reach that rank?

The answer is positioning. On the terms linked to legacy technologies and their modernisation, there is almost nobody. Thousands

of companies declare themselves on the fashionable terms. Only a handful declare themselves on ours.

A ranking does not measure quality. It measures the density of competition on a term. Choose the term.

The mistake to avoid

A warning to finish, because I have seen it ruin people.

Do not set up a company to change your social status. Set it up to capture value you already produce and that somebody else is collecting.

If you do not yet have a sellable skill, the structure will only cost you money and administrative time.

If you have a sellable skill but no channel to sell it, the structure will not solve the channel problem. It is necessary, it is not sufficient.

The sequence that works is: skill, then proof, then first client, then structure, then scale. Not the reverse.

> **THE TEST BEFORE YOU INCORPORATE**
>
> Ask yourself these three questions.
>
> **One.** Have I already been paid, at least once, for what I intend to sell?
>
> **Two.** Can I name three specific organisations that have this problem today, and explain why they cannot solve it?
>
> **Three.** Can I fund twelve months of fixed costs with no revenue from this company?
>
> Three yeses: incorporate. One no: prepare. Two nos: this is not the moment.

The next chapter covers the armour. That is, what turns an existing company into one a cautious buyer will sign with.

◆

CHAPTER 12

The armour

> Professional insurance, data protection, international transfers. What costs a few dozen euros a month and unlocks conversations that skill alone does not open.

A director is about to hand you a system his revenue depends on. He does not know you. You are in another country.

The question in his mind is not: can he code. It is: what happens if this goes wrong.

The armour answers that question before it is asked. It costs little. It changes everything.

Professional indemnity insurance

This is the centrepiece and the most underrated.

Professional indemnity insurance covers damage caused to the client in the course of the work. In software that means an intervention that causes an outage, data loss, lost revenue, or harm to a third party.

Real cost: a few dozen euros a month for cover that in my case runs into the millions.

Think about that ratio. For the price of a phone contract, you get a guarantee that covers a potentially fatal claim.

Cautious buyers are not buying the insurance. They are buying the fact that you judged it necessary to have some.

That is the decisive psychological point. The certificate is a signal. It says: this person anticipated the worst, assessed the risk, paid to cover it. They are serious.

A supplier with no insurance implicitly says the opposite.

> **ON THE GROUND**
>
> In commercial conversations, insurance rarely comes up head on. It appears in the footer, in the presentation document, in the terms and conditions.
>
> The other party almost never comments on it. They see it.
>
> The difference shows in the silence: questions about reliability stop being asked.

An important comparison. In some countries this kind of cover does not exist for an independent worker, or does not reach limits that are credible to an international buyer. That is precisely one of the reasons the foreign structure has operational value, not just tax value. The day equivalent insurance is available locally, the question will be worth revisiting.

Data protection

The second piece, and it has become central.

Every software engagement touches data. Even when that is not the purpose, you access databases, logs, backups, test environments full of real data.

Three things to put in place.

Registration with the relevant authority. In several jurisdictions, an entity processing personal data must register with the regulator. The process is online, the annual cost is small, and the number obtained is public and verifiable.

The data processing agreement. A contractual document setting out what you process, why, how, for how long, with what security measures, and what happens at the end. It is attached to the main contract.

Transfer analysis. This is the most technical point and the least well handled. As soon as data leaves the original jurisdiction, you must document the mechanism that allows it and assess the associated risk.

> THE TRANSFER QUESTION, IN THREE STEPS
>
> **One.** Does the regulation of the original jurisdiction apply to this processing?
>
> **Two.** Does the data leave that jurisdiction?
>
> **Three.** Is the recipient a separate legal entity located elsewhere?
>
> Three yeses: you are in a restricted transfer. You need a contractual mechanism and a documented risk assessment.
>
> This is not a question of where the server physically sits. It is a question of who controls the data and under which law.

The artificial intelligence case

This deserves developing, because it is the number one blocker with clients holding old systems.

You want to use a language model to analyse business code. That code contains:

- real data hard coded into the processing;
- table, schema and field names that describe the business;
- business rules that constitute the company's trade secrets;
- sometimes credentials and infrastructure paths.

A bank, an insurer or a logistics firm will refuse to let that out. Even with a contract guaranteeing no retention and no training.

That refusal is not irrational. It is prudent.

The answer is the airlock: mask locally before it leaves, process outside, unmask locally on return. Chapter 22 is entirely devoted to it.

What matters here is commercial. This is not only a technical device, it is a sales argument. You are not telling the client to trust a model provider. You are telling him that nothing identifiable leaves his systems, whatever his level of trust in that provider.

You are not selling artificial intelligence. You are selling the right to use it without losing control.

The documents that reassure

In practice, here are the documents I keep ready and that circulate before a signature.

- **Certificate of incorporation.** Verifiable on the public register.
- **Insurance certificate.** With the detail of cover, available on request.
- **Data protection registration number.** Verifiable online.
- **Standard confidentiality agreement.** Ready to sign, in both directions.
- **Data processing agreement.** Attached to the contract.
- **Service contract.** Scope, deliverables, intellectual property, reversibility, termination.
- **Security architecture note.** One page, explaining where the data sits, who accesses it, how.

These documents are prepared once. They then serve every client, with a few adaptations.

ON THE GROUND

Drafting these documents is now possible without a law firm, provided you are methodical.

The method I apply: I have the applicable texts researched, I demand primary sources, I have the official documents downloaded, I have several independent sources cross checked, and I demand precise references.

After that process I treat the document as reliable. That is not laziness, it is a multi pass check that few firms carry out as systematically.

On a major matter, professional advice remains essential. On routine documentation, this method is sufficient and free.

What the armour does not protect

We should be honest about the limits.

It does not protect you from a client who does not pay. That is the most frequent risk. No insurance covers it. Protection is contractual: deposit, milestones, reversibility conditional on payment.

It does not protect you from an expensive dispute. Even when you are right, proceedings cost time, energy and money. The best protection remains selection upstream.

It does not protect you from a failing partner. If you rely on a supplier, their failure becomes yours in the eyes of your client.

It does not protect you from a supplier's unilateral decision. A service can suspend your account, a provider can end a hosting arrangement, a platform can change its terms. I have lived through each of those.

> **RULE**
>
> For every critical dependency, have a fallback identified before you need it.
>
> Registered office, bank, hosting, email sending, domain name, model provider. Six dependencies, six fallbacks. The day one of them fails, you switch in hours instead of weeks.

The return on investment

Let us add it up.

The full armour costs, annually, the equivalent of a few days of work. It is a modest fixed cost.

What it unlocks: the ability to sign with organisations that, without it, would not have opened the discussion at all. Serious buyers have supplier onboarding procedures, and those procedures have boxes. Without the boxes you do not exist, whatever your skill.

The armour does not make you better. It makes you eligible. That is a different matter, and it is the one that counts.

The next chapter covers the roughest part of the whole arrangement: actually getting paid.

◆

CHAPTER 13

Money, currency and friction

> Getting paid from abroad when you live in a country with currency controls. The blocks, the rejections, the mediation, and the method that eventually gets things moving.

This is the chapter I wish I had read before I started.

Everything else is simpler than advertised. Setting up a company is a formality. Compliance is methodical paperwork. Prospecting is a mechanism.

Money is not. Money is hard.

The underlying problem

You live in a country where foreign currency is a constrained resource and where international movements are regulated. You run a company registered elsewhere. Your clients pay in foreign currency.

Three questions arise, and they do not have the same answers.

Where does the company get paid? Into a business account opened in the jurisdiction of incorporation, or with an international payment institution.

How is the director paid? Through the mechanisms provided by the applicable law, respecting tax residence rules.

What is brought back into the country of residence, and how? That is the most delicate question, and it requires following exchange control rules to the letter.

ABSOLUTE RULE

On this subject, never approximate, never work around, never improvise.

An aggressive tax arrangement or an undeclared flow exposes you to a risk out of all proportion to the gain. You are playing with two administrations at once.

Pay what you owe, declare what you receive, document every flow. The peace of mind is worth far more than the margin points lost.

The local status, the base of the whole arrangement

Before the foreign structure, you need an independent status in your country of residence.

That is what lets you invoice legally, here, as a person. It is the base that makes the rest explainable: the foreign structure is the commercial vehicle, the local status is the resident's tax base.

Opening that status, in my case, was simpler than the stories suggest. A business bank account opened the same day, in a neighbourhood branch, with no special connection and no favours. Online management subscription immediately after. It works.

I mention it because the opposite belief is widespread, and it discourages people who would have every chance.

A large share of the barriers we are told about do not exist. They are described by people who never tried.

International banking friction

Here is the unpleasant part.

International payment institutions apply automated compliance policies. Those policies classify whole jurisdictions by risk criteria, and trigger enhanced checks on entirely ordinary flows.

I had a client deposit blocked. The transfer was compliant, the documentation complete, the origin of funds documented, the client identified. It was rejected.

What followed is instructive.

First line support has no decision making power. It follows a procedure and repeats the same answer. Delays stretch out. The relevant mediation service, in some countries, is itself so overloaded that its public reputation is dreadful.

What finally unblocked the situation: a public, factual, documented review on a consumer review platform. No insults, no exaggeration, with the facts, the dates and the references.

Within days a compliance manager took over the file personally, approved it, and the original decision was reversed.

WHAT THIS EPISODE TEACHES

One. In a large organisation, the official channel is often the slowest. The reputational channel is often the fastest.

Two. Use it with absolute rigour. Verifiable facts only, never a personal attack, never an exaggeration. A factual review is protected. An accusation is not.

Three. The cost of handling that incident, for the institution, very probably exceeded the amount of the transfer. That is absurd and that is how it is.

Four. Never depend on a single institution.

The three account rule

This is the most important operational lesson in the chapter.

Open several business accounts from the start, while everything is going well and you do not need them. That is the only time they are granted easily.

A blocked account means a stopped business. You can no longer get paid, pay your suppliers, or renew your critical subscriptions. Your services cut out one after another.

I have three. It requires administrative discipline and closer bookkeeping, and it is worth every minute.

> **RULE**
>
> Three accounts minimum, with three different kinds of provider: a conventional bank in the jurisdiction, an international payment institution, a second backup payment institution.
>
> Put a real transaction through each one, regularly. A dormant account is an account that will be closed or frozen at the worst possible moment.

Payment cards, an underrated tool

A practical point with more impact than it looks.

A company can generate virtual payment cards, capped, time limited, dedicated to one use.

What that makes possible in practice:

- subscribing to online services without exposing the main account;
- activating trial periods that require a card, with a cap of a few euros;
- assigning a card to one purpose and freezing it afterwards;
- giving access to a colleague or a partner without granting them rights.

It is a modest building block that unlocks a great deal, particularly access to the programmes and trials described in the next chapter.

Invoicing terms

What works, in my experience.

The deposit. Systematic above a certain amount. It is not there to fund the work, it is there to filter. A client who refuses any deposit is telling you something.

Milestones. Payment follows deliverables, not time. That aligns interests and protects both sides.

Conditional reversibility. Full transfer of access and ownership happens on final payment. That is not holding anyone hostage, it is a standard clause, and it must be written from the start.

Refusing promises. A share of equity later, a percentage of future revenue, a subscription once the platform works. I have had those proposals several times.

When someone offers to pay you later with a success that does not exist yet, they are asking you to invest without giving you shares.

My answer has become simple: if the project is good, it finds funding. If it cannot find funding, it is not my job to carry it.

There is one exception, and it is strategic: work given deliberately, for a clearly defined and clearly obtained non financial return. That is the subject of chapter 16. But it is an investment decision, not a concession extracted in negotiation.

Your relationship with money

I will finish with something less technical and more decisive.

Money comes in and goes out. What matters is the relationship you have with it, because it drives your decisions far more than

your balance does.

Two stances exist.

The first is scarcity. Everything is a threat, every spend is a loss, every share is a risk. It leads to accumulating and doing nothing with what you accumulate.

The second is abundance. You invest in people, you share resources, you accept delayed or zero returns.

I chose the second, without being naive. In practice that means I pay discretionary bonuses when work deserves it, I share the subscriptions and credits I have, I pay for knowledge transfer sessions with strong students to stay current, and I give away complete deliverables when I judge the exchange to be good.

That is not disinterested generosity. It is a long term calculation. But you need the cash to afford it, and you need to know when to stop if the return never comes.

> **RULE**
>
> Never seek to own what you can rent.
>
> A vehicle, premises, heavy equipment tie up capital and create fixed costs. A service can be plugged in, unplugged and replaced.
>
> That discipline frees up cash for the only thing that actually returns anything: the ability to seize an opportunity when it passes.

The next chapter shows how far this logic can be pushed. How to obtain industrial grade infrastructure and tooling at a cost that approaches zero.

◆

CHAPTER 14

Almost no cost at all

> Tens of thousands of euros in infrastructure credits, professional tools, research access. How you get them, what they ask in return, and the trap they contain.

Here is one of the most profitable imbalances in today's market, and it is massively under used.

The large infrastructure and tooling providers are fighting for market share. Their marginal cost is low, their goal is adoption. So they hand out credits, licences and access on absurd terms, betting on your future dependency.

That future dependency is the trap. The credit itself is a real opportunity.

They are buying your dependency. Sell it to them, but keep the receipt and the exit door.

What is actually available

Without naming providers, here are the categories.

Startup programmes from the large infrastructure providers. They hand out credits in tiers. A first slice on entry, a larger second one after a file describing the project is reviewed, a third after real usage is demonstrated.

In my case the progression was exactly that: a first allocation on registration, then a much larger one after review of the modernisation engine project, with the prospect of a further tier by keeping resources running.

Developer platform programmes. Access to professional tools, continuous integration environments, development assistants, with generous quotas.

Deal aggregation platforms for founders. They gather discounts and free access to dozens of services. The combined value is considerable.

Academic and documentary access. National research portals give access to scientific databases whose individual subscriptions are out of reach. That access is valuable when you work on sharp technical subjects.

Trial periods. Taken alone they are nothing. Organised and planned, they cover a whole phase of development.

> **THE LEVEL REACHED**
>
> By combining programmes, a new company can gather tens of thousands of units of infrastructure credit over a year, plus a full professional toolkit.
>
> In practice that means several running servers, managed databases, high volume artificial intelligence services, continuous integration pipelines, storage and monitoring.
>
> For a business of one to three people, that is far better infrastructure than many established companies allow themselves.

The real conditions

Nothing is free. Here is what is asked.

A legal entity. Most programmes require a company registered in an eligible jurisdiction. That is the first return on the structure described in chapter 11.

A credible file. A description of the project, the target, the intended use of the infrastructure. This is not a competition, it is an anti abuse filter. An honest, precise file gets through.

A card on file. Hence the value of capped virtual cards.

Usage milestones. Higher tiers are conditional on real usage. In practice that means keeping resources running for a given period.

That is where you have to be clever. Keeping resources running consumes credit. So you deliberately allocate part of the current credit to unlock the next tier. That is a calculated investment, not a cost.

> RULE
>
> Treat these programmes as a budget line, with a calendar.
>
> Note the expiry dates. Note the milestones. Schedule heavy compute work inside the windows where credit is available.
>
> A credit that expires unused is a pure loss.

The dependency trap

Now the warning, because it is serious.

These programmes are designed to create dependency. You build on one provider's services, you use its proprietary compon-

ents, and the day the credits run out, migration costs more than the invoice.

That is a perfectly rational model on their side. It only becomes a problem if you fail to anticipate it.

My discipline comes down to four rules.

One. Everything must be containerised. An application packaged in a container moves from one provider to another in a few hours.

Two. No proprietary component on the critical path. Managed, provider specific services are seductive and confining. A standard database exports. A proprietary service does not.

Three. A simple server as the reference. I always keep in mind what a modest machine can absorb. That prevents oversizing out of comfort and keeps a realistic exit route.

Four. An up to date estimate of the real cost. What happens the day the credit disappears? If the answer is I do not know, you have a latent problem.

ON THE GROUND

I have seen a technical team recommend a complex distributed architecture to a director, with container orchestration and a split into microservices, for a platform whose real load fitted very comfortably on a modest server.

That practice has a name in the trade: curriculum driven development. You choose the technology that will look good on your profile, not the one that solves the problem.

I have seen applications serve very large request volumes on a machine with a few gigabytes of memory, properly configured.

Saying that in a room full of experts does not make you popular. It is nonetheless the most useful piece of information the director will hear that day.

The effect on your cost base

Look at what this actually changes.

Classic services company model: the main cost is people. Salaries, contributions, premises, supervision. Infrastructure is a secondary but real line.

Model with methodical use of programmes: infrastructure tends towards zero during the start up phase. The cost of intelligent assistance, which would otherwise be significant, is absorbed by the credits.

What remains is human time. That is the only real cost.

When infrastructure costs nothing and tooling is given away, the only remaining variable is time. The whole strategy is then to protect it.

What this makes possible in practice

Three uses I have genuinely put into practice.

Giving away a complete deliverable to open a door. When building a platform costs a few hours of your time and zero euros of infrastructure, you can give it to a strategic contact. The gap between the value he perceives and your real cost is the best commercial lever there is.

Equipping the people you work with. Interns, partners, former colleagues. Access to professional tools changes a student's path, and costs you only a seat in a licence you already hold.

Absorbing an exceptional workload. Analysing a huge volume of code, processing thousands of documents, running a full campaign. Those workloads would be prohibitive at full price. They become trivial decisions.

THE OPPOSITE TRAP

There is a symmetrical risk: uncontrolled consumption.

The most powerful models consume quickly. I have seen large credit balances drain in a few days during intensive work, with several sessions running in parallel.

Set caps, alerts and per project tracking. Otherwise you will discover the exhaustion at the worst moment, in the middle of a delivery.

The general lesson

This abundance of tools is a market anomaly. It will not last forever.

What will remain is what you built while it lasted: clients, a reputation, software assets, a method.

Those who use these resources to produce something durable will come out with a considerable lead. Those who use them only to consume will be back at zero the day the tap closes.

RULE

Free credit does not build a business. It funds the window in which you must build one.

Treat every credit as a countdown, not as an income.

The next chapter covers the resource that changes the most: intelligent assistance treated not as a tool, but as a team.

CHAPTER 15

Artificial intelligence as a team

> Neither magic nor threat. A multiplier that only amplifies those who already know where to look. How I really use it, and where it fails.

I will be blunt, because the subject is drowning in noise.

Artificial intelligence does not replace an engineer. It multiplies the one who already knows what he wants to obtain. On someone who does not know, it multiplies by zero, and the result is code that works in a demo and collapses in production.

It has never invented anything for me. It has carried out, at scale, approaches I designed.

What I actually use it for

Here is my real usage, in order of value produced.

Archaeology. The first use and the most profitable. Understanding a system nobody understands any more. Mapping it, extracting business rules, identifying customer branches, spotting dead code. That is heavy reading work, repetitive, demanding on attention. It is exactly what a machine does better than a tired human.

Repetitive structural work. Writing data access layers, validation schemas, tests, technical documentation, migrations. That is

not where an engineer's value sits, and yet it is where most of his time goes.

Producing documents. Presentations, audit reports, architecture notes, commercial proposals. A fifty page document with diagrams can be produced in minutes from a properly assembled knowledge base.

Legal and regulatory research. With the method described in chapter 12: primary sources demanded, cross checking compulsory, verifiable references.

Speeding up discovery. Taking over an unknown code base, understanding a framework you have never used, producing a first working prototype in a technology you barely know.

> **A CONCRETE EXAMPLE**
>
> A partner sent me more than a hundred and forty pages of functional documentation on an old proprietary application, produced by hand, the result of considerable work.
>
> Reproducing the equivalent in a modern cross platform technology took half a day, even though I am not a specialist in that mobile technology. Tens of thousands of lines that work.
>
> The value was not in the generating. It was in knowing how a project is structured, which architectural patterns to use, what to ask for and in what order.
>
> Without that knowledge, the same tool would have produced an impressive and unusable prototype.

The method that works

A few hard won principles.

Never start from a blank page. Provide the context: the structure of the existing project, the conventions, the constraints, examples. The quality of the result is proportional to the quality of the context you give.

Break it down. Do not ask for an application. Ask for a module, with defined inputs and outputs. Then the next one.

Demand verification. Have the tests written, have them run, have the failures fixed. A chain that checks itself produces a completely different level of reliability.

Work in parallel. Several sessions on several subjects at once. That is where the gain becomes genuinely large. While one module is being built, another is being reviewed, a third documented.

Never ship without reading. That is the non negotiable rule. You remain responsible. Your insurance covers your liability, not the model's.

> RULE
>
> The limiting factor is not the power of the model. It is your ability to state what you want, break the problem down and check the result.
>
> Those three skills are exactly those of a good software architect. Artificial intelligence has not devalued that role, it has made it decisive.

The autonomous agent myth

A word on a very common claim.

You regularly see the idea that an autonomous agent, set loose on a task, will produce a complete application on its own while its owner does something else.

That is true for the visible part. It is false for everything else.

What is systematically missing from those autonomous outputs:

- rate limiting, so an open door onto an uncontrolled bill;
- secrets management, often left in plain text;
- terms of use, privacy policy, legal notices;
- data processing compliance;
- a deployment, backup and restore strategy;
- monitoring and alerting;
- reversibility.

In other words: everything that separates a demo from a usable product.

An engineer with a broad view does in one afternoon what an agent left alone will never finish properly.

Cost control

A critical point, because it can destroy your cash.

An unprotected endpoint exposing an artificial intelligence service is an open invoice. All it takes is one bot finding the address.

Three compulsory protections:

Rate limiting per user, per address, per key, at realistic thresholds.

Spending caps at provider level, with alerts at intermediate points.

Per feature logging, so you know what is consuming.

Add selective retrieval: never send a whole corpus with every request. Index it, select the relevant fragments, send only those. The saving is measured in orders of magnitude.

> **WHAT OPTIMISATION CHANGES**
>
> On the same workload, between naive use and optimised use, I have seen cost differences of well over a factor of ten.
>
> The levers: choosing the model to suit the task, sending targeted context, caching stable answers, batching, and dropping to a cheaper model for simple work.
>
> That optimisation is itself a sellable service. Many organisations are paying ten times what they should.

Where it fails

Let us be precise about the limits, because that is where money is lost.

Rare proprietary languages. On the least publicly documented legacy technologies, models invent. They produce instructions that do not exist, with total confidence. That is the central problem of chapter 22 and the reason for the hybrid approach.

Implicit business context. A rule written down nowhere cannot be guessed. The model will produce something plausible and wrong.

Binding architectural decisions. It will always suggest the most common solution, not the most suitable one. On a structural choice, it is the worst possible adviser.

Anything requiring judgement. Should I take this client? Should I ship now? Should I tell this director his supplier is misleading him? No assistant will answer that.

The effect on the business model

The most uncomfortable point, and it has to be faced.

A services company invoices time. A fivefold productivity gain mechanically divides its revenue by five, at constant volume.

Three possible outcomes.

Say nothing. Many do. The gain is absorbed as internal comfort.

Move to outcome pricing. Invoice the deliverable, not the time. That is the only lasting outcome, and it requires a level of trust few commercial relationships reach.

Widen the scope. Take on what you did not take on before, because it was too expensive: recovering abandoned systems, documenting an existing estate, full audits.

I chose the third. It has a decisive advantage: it does not destroy perceived value and it opens a market nobody was serving, because it was economically out of reach before.

> **RULE**
>
> Do not sell your productivity gain by cutting your prices. Sell it by widening what you are able to take on.
>
> The first route turns you into a price competitor. The second turns you into a player in a market your competitors cannot serve.

Chapter 24 deals with the dark side of that speed: what happens politically when you deliver too fast.

But before you deliver, you have to sell. That is the subject of the whole next part, and it is the skill nobody taught you.

◆

PART FOUR

Winning work

The skill university does not teach, and the one that decides everything. Giving before asking, building a prospecting system that works without a network, making your own machine for finding clients, diagnosing before selling, and knowing how to refuse. Five entirely practical chapters.

CHAPTER 16

Give first

> Entering a closed market through the back door. Mentoring, free work, the free diagnosis. What it really opens, and the exact moment you must stop.

You have no network, no local references and no physical presence. You are aiming at a market where everyone knows everyone.

The front door is shut. A sales message into the void, a cold call ignored, a speculative application buried.

There is still the back door, and it is wide open: give before you ask.

You do not get in by asking. You get in by bringing something nobody had offered.

The mechanism

It is simple to state and demanding to carry out.

You find places where directors are looking for help. Mentoring platforms, business support networks, professional communities, sector associations. They exist in every country, many are backed by institutions, and they are permanently looking for qualified contributors.

You register as a mentor. You give time, free, with no sales agenda.

What that produces, in order:

One. You get direct access to directors. Not to their assistants, not to a procurement department. To them.

Two. You see their real problems. What a director tells a mentor bears no relation to what he tells a supplier. You gain market knowledge no study would give you.

Three. You are seen as a peer, not as a salesperson. That is a radical change of position.

Four. You accumulate written, public, verifiable recommendations. That is the currency of credibility for someone with no local references.

What I actually do in a session

Here is the precise mechanism, because that is where it all happens.

Before every session, I prepare. Genuinely. I look at the person's activity, their website, their platforms, whatever is publicly visible of their infrastructure, their legal and filing situation on public registers.

Then I produce a complete document: context, the existing architecture as I read it, the weak points I identify, a proposed target architecture, verified costs, concrete steps. Thirty to fifty pages with diagrams.

That document is given away. It is never invoiced.

THE EFFECT IT PRODUCES

A director who receives an analysis of his own system, done by someone he has not paid, with real numbers and workable solutions, is having an experience he has never had before.

His usual suppliers sell him proposals. Nobody had ever explained his own system to him.

That document costs one or two hours thanks to the tooling described in chapter 15. To him it is worth several days of consultancy.

The gap between the real cost and the perceived value is the best commercial lever there is.

I have gone further on occasion: arriving at a session with a working prototype already deployed on a subdomain, with a certificate and access. Fifteen minutes of extra work.

The effect is striking. You do not explain what is possible. You show it running.

What it actually produced

Let us be factual.

This approach gave me access to directors of established companies, to commercial directors used to growing multi million pound businesses, to former heads of large public organisations who had become coaches, to founders of independent firms.

It earned me written public recommendations from people who are credible in their market.

It led to my being selected to run a masterclass for an entire support ecosystem covering tens of thousands of supported organisations and thousands of mentors.

It got me publicly shared by highly visible people, which is exposure no reasonable advertising budget could buy.

And it gave me something more important: intimate knowledge of the real problems of a market I had never visited.

You can spend your life mentoring one person at a time. Being invited to speak to a room full of them is another matter, and you do not ask for it. You earn it.

The content that works

A word on that masterclass, because it shows what actually holds a room of directors.

Five subjects, none of them a technical demonstration.

One. The reality of business systems nobody talks about. How organisations really run today, on premises, hybrid, in the cloud, at international scale. Why huge volumes of old code are still running, why even large public bodies only started migrating very recently, and why rewrites fail.

Two. Artificial intelligence in production, concretely. Real integration patterns, with rate limiting, security and direct access to the interfaces. You use it, you do not reinvent it, and you put guard rails around it so that one unprotected endpoint never blows up a bill.

Three. Architectures for the real world. On premises against cloud against hybrid, and data sovereignty handled seriously.

Four. Building with first class tools at almost no cost. Credits, programmes, licences. How a founder can run a real platform for very little while validating the market.

Five. Vendor lock-in and how to avoid it. Designing for independence from day one.

What surprised me most in the feedback: the appetite for substance. Those directors did not want magic buttons. They wanted to understand how the machine really works, so they could decide for themselves.

> RULE
>
> Never talk technology to a director. Talk risk, deadline, cost and dependency.
>
> Technology is what you do. It is not what you sell.

The limits, and they are real

Now the other side, because this chapter would be dishonest without it.

Free work creates an expectation, and sometimes an abuse.

I built, free of charge and in a few days, a complete digital ecosystem for a not for profit organisation in a foreign country. A matching platform, session management, profile management, certificates, data protection, the whole apparatus.

The agreed return was not financial: a seat on the board, an introduction to their network, some visibility. It was an explicit exchange.

The result: the platform delivered, the return never honoured. No recommendation, no introduction, no follow up. They were not even able to deploy it themselves.

I ended the collaboration.

> **TRAP**
>
> When a non financial return is not dated and written down, it does not exist.
>
> You do not have to invoice. You do have to define what you expect, in what form, and by when.
>
> An exchange with no deadline is not an exchange, it is a gift. Make gifts when you decide to, not when they are extracted from you.

A second limit, subtler. What you give becomes the expected standard.

I gave a great deal over three years: supervising interns on my own time, free public sessions, community projects, supporting students. The day I cut back, I did not return to neutral. I became the one who stopped.

That is mechanical and you have to accept it in advance.

The selection rule

After several experiences, here is how I decide.

I give when:

- the person has real decision making power in their organisation;
- they operate in an ecosystem I want access to;

- the effort costs me hours, not weeks;
- what I produce is also useful elsewhere, as a demonstration or a reusable component.

I do not give when:

- the person wants a full build with no commitment;
- the return is an undated future promise;
- the scope grows with every exchange;
- I sense my time is not respected, which shows from the first meetings.

> **THE SIGNAL THAT NEVER LIES**
>
> Watch how the person treats time before there is any money involved.
>
> Someone who moves a session he set himself, who arrives late without apology, who cuts a meeting short for another call, is showing you exactly how he will behave once the contract is signed.
>
> I have had those signals several times. Every time I ignored them, I paid for it.

The switch

One last point about timing, because it is decisive.

Giving first is an entry strategy, not a business model. It has a shelf life.

Its purpose is to move you from unknown to known, from unreferenced to recommended, from outsider to insider. Once those

three transitions are done, it must give way to a normal commercial relationship.

You recognise the moment by a simple sign: people start approaching you without your having offered anything.

The day you stop chasing opportunities, stop working for free. That is the moment to invoice.

The next chapter sets out the full mechanics of prospecting: how to go and find a client when you know nobody.

◆

CHAPTER 17

How prospecting actually works

> How to go from a hundred thousand theoretical prospects to four hundred real targets, and why a free diagnosis beats any sales pitch.

This is the most useful chapter in the book. It is also the one whose content is taught on no engineering course.

The starting reversal

Classic prospecting means talking about yourself to people who did not ask. It does not work, and it works less and less.

Prospecting that works means talking about them, with information they do not have, and demonstrating a skill before offering it.

Do not sell what you can do. Show what you have already seen at their end.

Step one: define the real target

Most beginners define their target like this: companies that need development work.

That is unusable. A target is defined by criteria you can observe and filter on.

Here is how I defined mine.

Sector. Organisations whose business depends on an old and critical information system. Distribution, logistics, mid sized financial services, healthcare, industry.

Size. Large enough to have a budget and a real dependency, small enough not to interest the large integrators. That is the blind spot described in chapter 7.

Technical signals. Old technologies detectable from outside: end of life language versions, obsolete frameworks, server headers, proprietary technologies visible in job adverts.

Human signals. This is the most predictive criterion and the most neglected. A job advert open for a long time on a rare technology means they cannot find anyone. A technical profile in the team showing as open to opportunities means a departure is coming. A technical manager whose public activity suggests exhaustion means an organisation under strain.

Regulatory signals. A compliance deadline, an audit, a new obligation. An external constraint creates a budget where there was none.

> **RULE**
>
> The best buying signal is not technical, it is human.
>
> An organisation does not modernise because its code is old. It has put up with it for twenty years. It modernises because the person who keeps it running is leaving.
>
> Look for the departure, not the code.

Step two: cut down hard

Here is the real calculation I made.

An advanced professional search tool gives access to a pool of tens of thousands of organisations loosely matching a sector and a size.

By applying precise boolean queries, combining sector, size, contact role, technical signals and human signals, that pool falls to a few hundred.

Four hundred real targets instead of a hundred thousand theoretical ones.

That is where all the work happens. Prospecting the four hundred works. The same prospecting across the hundred thousand fails, costs money and damages your reputation.

> **THE CONVERSION RATE**
>
> On a campaign run at that level of filtering, I saw a conversion rate far above what is usually quoted for cold outreach.
>
> Two explanations, and neither is magic.
>
> **One.** The filtering. When the market is narrow and the skill is rare, every relevant contact is close to the need.
>
> **Two.** The free audit sent up front. I do not ask for a meeting. I send a finding.

Step three: the message

This is the part everyone gets wrong.

A prospecting message that works has a strict structure.

The opening is a fact observed at their end. Not a presentation of you. A precise, verifiable finding that proves you looked.

The body turns that fact into a business risk. Not a technical problem. A director does not react to an obsolete version. He reacts to an outage, a regulatory exposure, a dependency on one person.

The offer is asymmetric. You offer something concrete and bounded. Not a discovery call: a deliverable.

One single call to action. A closed question that can be answered yes or no.

> **WHAT KILLS A MESSAGE**
>
> **Head on criticism.** I wrote to an organisation whose platform had visible weaknesses. I described the problems then offered to fix them. Total silence.
>
> The mistake: I had criticised before offering. In the same message, the reader becomes defensive at the first sentence and never reads the second.
>
> The correct wording links the finding and the solution in the same breath, with no judgement on whoever built the system. That person may be reading the message.
>
> **Generality.** Any message that could be sent to a thousand recipients is identified as such in three seconds.
>
> **Length.** A director reads on his phone between two meetings. Beyond a few lines, he archives it.

Step four: the sequence

A single message is almost never enough. A spaced sequence works.

First contact. The finding and the free deliverable.

Follow up a few days later. Short. You add something new, never a repetition. A second observed signal, a clarification, a sector reference.

Final follow up. Shorter still. It closes cleanly: I remain available, I will stop contacting you.

That last message produces a surprisingly high reply rate. It releases the other person from the implicit obligation, and many reply at that point.

> RULE
>
> Never follow up by repeating. Follow up by adding.
>
> Every message must contain something the previous one did not. Otherwise you are not persistent, you are pushy, and those are opposite perceptions.

Step five: the legal frame

A serious point, because it decides whether your approach is solid or fragile.

Business to business prospecting is regulated, and the rules differ by jurisdiction. What is allowed in one country is not allowed in another.

Three principles I apply.

One. Strict professional relevance. The message must relate to the recipient's professional activity, within their role.

Two. Full identification. Who is writing, from which entity, with verifiable contact details. No disposable sender addresses, no vague identity.

Three. Immediate and respected opt out. An unsubscribe request is honoured without delay and permanently.

> **THE LINE YOU DO NOT CROSS**
>
> Sending a technical assessment of a system you were not authorised to examine is a sensitive area.
>
> My rule: only publicly accessible information, obtained without any circumvention, with no intrusive testing, no attempt at access. You observe what is exposed, you probe nothing.
>
> That distinction is fundamental. On one side it is open source analysis. On the other it is a criminal offence.
>
> Stay comfortably on the right side. The grey area has no economic interest and a major legal risk.

Step six: useful disappointment

A word on what prospecting teaches you, independently of any sales.

It teaches you who you are on the market.

I discovered that my offer was poorly aligned with a market I knew very well, and well aligned with one where I knew nobody. That information is worth more than the first contracts.

I also discovered that a large share of global prospecting volume is very low quality. I receive around a dozen approaches a day from foreign agencies. They all look the same. They persist, so it works at some rate.

If persistent mediocrity gets results, persistent quality will get more. The variable that matters is persistence, not talent.

What to remember

Six steps, in order:

1. Define the target by observable criteria, not by intentions.
2. Cut down hard, to a few hundred.
3. Write a message built on a fact observed at their end.
4. Sequence by adding, never by repeating.
5. Stay comfortably inside the legal frame.
6. Treat refusals as market information.

The next chapter shows how to industrialise all of this, by building your own machine rather than renting somebody else's.

◆

CHAPTER 18

Building your own machine

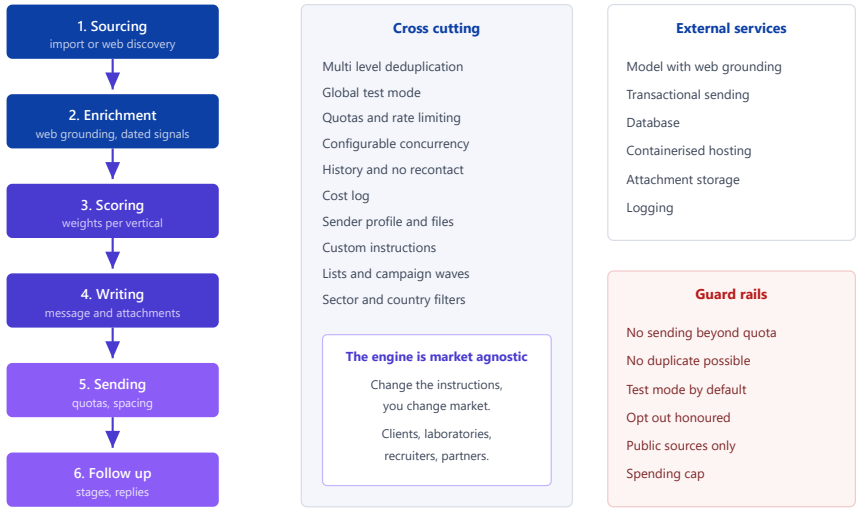
> The full architecture of a prospecting engine: sourcing, enrichment, scoring, writing, sending, following up. And why the same engine finds you a client, a research laboratory or an internship.

Commercial tools exist for all of this. They are expensive, they impose their own logic, and above all they take away the only thing that matters: control of the selection criteria.

I built my own. Here is the full architecture.

Renting a machine means accepting somebody else's definition of your ideal client.

Overview



Functional architecture of the prospecting engine.

Step 1. Sourcing

Two possible inputs, and both are useful.

Import. You already have a list, built by hand, pulled from a directory, or received from a partner. The engine takes it in, normalises it, deduplicates it.

Discovery. This is the powerful part. You describe what you are looking for: target country, sector, type of organisation, themes, size. The engine queries a model with web grounding, meaning one that can actually go and read pages.

It does not merely search. It goes into the sites, reads the team pages, the legal notices, the job adverts, the publications. From that it extracts organisations, people, roles and public professional contact details.

You can run a discovery across several hundred targets in one operation. The cost per contact discovered is a few cents.

> **WHAT THAT CHANGES**
>
> The difference between a classic search and web grounding is like the difference between reading a directory and sending someone to visit the premises.
>
> The engine brings back things no commercial database contains: a project mentioned on an internal page, a collaboration cited in a publication, a technology visible in a job advert.
>
> That material is what makes a personalised message credible, because it is true and verifiable.

Step 2. Enrichment

For each contact, the engine builds a record: organisation, role, themes, recent work, visible technologies, and above all dated signals with their source.

The absolute rule: every signal must come with its source. Without a source it is discarded. That is what prevents invention.

Step 3. Scoring

Each contact gets a score and a grade, from weighted criteria you define.

The criteria vary with the market. For commercial prospecting: sector fit, size, technical signals, human signals, accessibility of the contact. For academic outreach: closeness of subject, recent activity, past supervision, affiliation.

The crucial point: the weights can be changed and the whole portfolio can be rescored. You adjust your definition of the ideal target and recalculate everything.

> **RULE**
>
> Scoring is not there to rank. It is there to eliminate.
>
> Its job is to stop you writing to people who have no reason to reply. An engine that brings back three hundred contacts and discards two hundred earns you more than one that brings back a thousand.

Step 4. Writing

The engine produces the message from three sources:

- your full profile, injected automatically: background, expertise, what you are looking for, constraints, duration, available attachments;
- the signals actually found on that specific contact;
- your general instructions and, if needed, a specific instruction per contact or per list.

The format is constrained: an opening based on a real signal, a value proposition with no jargon, one call to action, a signature.

Attachments can be adapted to the recipient. On that point I am cautious: generating an entirely synthetic document is a bad idea. Selecting the right version of an existing document based on the recipient's profile, on the other hand, is entirely legitimate.

A REAL EXAMPLE OF OUTPUT

A message generated for an academic contact mentioned his precise work, his latest publication, and connected it to the sender's background.

It also mentioned a past collaboration with a university in a specific city. That information had never been given to the system. It was accurate: the engine had found it while exploring the laboratory's pages.

That is exactly the level of personalisation that triggers a reply. No generic template produces that.

Step 5. Sending

Three structural choices.

The sending provider. Either a dedicated transactional service on your own domain, or the interface of your professional mailbox. The second gives better deliverability, with stricter quotas.

Quotas. Aligned with the real limits of the service. Going over does not save time, it loses you the account.

Test mode. On by default. All sending is diverted to a control address, with a visible banner in the interface.

THE GUARD RAIL THAT SAVES YOU

Test mode is not a development convenience. It is what stops you sending several hundred badly formed messages to real contacts, with no way back.

A contact address is only burned once.

Only switch it off after reading at least twenty generated messages in full, attachments included.

Step 6. Follow up and tracking

Contacts move through stages: new, analysed, contacted, followed up, replied, closed.

The follow up is generated from the initial message, shorter, with a reply subject line. It can be produced in bulk and then sent in bulk.

An honest limitation: without read access to the mailbox, the system cannot detect replies automatically. Moving to the replied stage is manual. That is a deliberate trade off, because read access raises a confidentiality question I prefer to avoid.

Deduplication, in detail

This is the most underrated component and the one that took the most reworking.

Three levels:

Level 1. Exact address, case insensitive.

Level 2. Fuzzy match on the normalised name. Accents ignored, initials ignored, word order ignored. The same person appearing

under two different forms with two different addresses is caught as a duplicate.

Level 3. Organisation and role pair.

On top of that sits a history: you do not recontact someone before a set delay, even if the campaign changes.

Contacting the same person twice in the same week destroys a reputation that took months to build.

The most important point

The engine is market agnostic.

The same system, with different instructions, will:

- prospect organisations for a service offer;
- identify research laboratories abroad for a final year project;
- contact recruiters about a rare skill;
- look for partners or distributors.

I adapted the engine in one evening for academic internship search. A student was able to contact hundreds of researchers worldwide, each with a message built on their real work.

THE REAL RUNNING COST

Discovery and enrichment: a few cents per contact.

Writing: negligible per message.

Sending: free within the quotas of a professional mailbox, or a few euros a month through a transactional service.

Hosting: a modest server.

A full campaign across several hundred contacts, entirely personalised, costs less than a lunch.

That asymmetry is what makes building preferable to renting.

The product inside the product

One last remark, and it is strategic.

A tool built for your own use is very often sellable. The need you solved, thousands of others have too.

You do have to be clear eyed: a prospecting tool sold to the general public feeds exactly the noise this chapter explains does not work. The value is not in mass sending, it is in filtering.

If you commercialise this kind of engine, sell the filtering, not the volume.

The next chapter covers the deliverable that turns a contact into a client: the diagnosis.

◆

CHAPTER 19

Diagnosis before the sale

> Why an audit given away wins more contracts than any proposal. What goes into it, how you produce it quickly, and where the trap is.

There is a moment in every commercial relationship where the balance tips. The other person stops seeing a supplier and starts seeing someone who can help.

That moment almost never comes from a presentation. It comes from a finding.

A proposal says what you could do. A diagnosis says what is happening. Only the second one is verifiable straight away.

Why it works

Put yourself in a director's position.

He receives sales messages every week. All of them say the same thing: we are experts, we have references, we can help you.

He has no way of telling them apart. So he ignores them all.

Then one message arrives that does not talk about the sender. It talks about him. It says: your platform runs an end of life version,

your certificate expires in three weeks, your job advert for that technology has been open since March, and here is what it costs you.

That message is not read the same way. It is verified.

And once it is verified, the relationship has changed. You are no longer someone asking for something. You are someone who saw something.

What a good diagnosis contains

A structure that has proved itself over several dozen documents.

One. The context, in their words. What the organisation does, who its customers are, what its system supports. That section demonstrates that you understood the business before looking at the technology.

Two. The architecture as it stands. What you observed, with the sources of each observation. A diagram is worth ten pages.

Three. The findings, in order of risk. Not in technical order. In business risk order. An expired certificate blocking sales comes before an outdated library.

Four. The estimated impact. This is the part most engineers skip and the one directors read. What does that finding cost, in outage, in exposure, in delay, in dependency.

Five. The target architecture. What it should look like, with real costs. Real ones. Not orders of magnitude.

Six. The route. Steps, in order, with what each one unlocks. And crucially: what can be done without touching the rest.

Seven. What I would not do. That section is the one that establishes credibility. Saying what is not worth doing proves you are not selling volume.

> **RULE**
>
> Every diagnosis must contain at least one recommendation that earns you nothing.
>
> Something they can do themselves, this week, for free. That single element does more for trust than the whole rest of the document.

How you produce it quickly

Without the tooling from chapter 15, this exercise would cost several days and would not be economic to give away.

With it, the sequence is as follows.

Public collection. Everything visible: site, exposed headers, public certificates, job adverts, legal notices, filed accounts, public technical activity. No probing, no attempt at access. Chapter 17 explains where the line sits.

Structuring. The collected material becomes a knowledge base. Every element keeps its source.

Analysis. That is where experience is decisive. The machine sees the signals. It does not know which one matters. A version number is a fact. Knowing that this version, in that sector, with that data, creates a regulatory exposure, is judgement.

Production. Document, diagrams, costing. A few hours.

Review. Full reading, every claim checked. This is not optional. A single wrong finding destroys the whole document.

THE TRAP THAT RUINS EVERYTHING

An assistant that invents a plausible finding is the worst risk of this exercise.

I have seen generated analyses assert an outdated version that was in fact current, a non existent vulnerability, a wrong cost.

If you send that, you lose everything at once. Not just the contract. Your credibility with someone who talks to other directors.

Rule: every factual claim carries a source you have personally verified. If you cannot verify it, remove it.

The right level of detail

A common mistake is to make it too technical.

The document is read by a director, not by an engineer. The engineer will read it afterwards, and he must find it rigorous, but the first reader decides.

That means:

- an executive summary of one page, readable in two minutes;
- findings phrased in business terms, technical detail in an appendix;
- diagrams instead of descriptions;
- costs in currency, deadlines in dates.

THE SENTENCE THAT WORKS

Not: your application uses an unsupported version of the runtime.

But: the software layer that runs your order taking has had no security updates for two years. One flaw published today would stay open. Your insurer may ask you about that.

Same fact. Two completely different reactions.

What happens next

The diagnosis is given away. That is the point. It is not a loss leader, it is a demonstration.

Three outcomes.

No reply. Frequent, and it is not a failure. Some organisations read, file it, and come back six months later when the situation forces them to.

A reply with questions. The best signal. It means the document was read and taken seriously. That conversation converts far more often than any meeting requested cold.

A reply asking for a proposal. Then the game changes. You are no longer competing on price against three suppliers. You are the one who identified the problem.

The one who names the problem sets the framework of the answer. That is the real advantage, and it is worth much more than the days you gave away.

When not to do it

Three cases where I do not produce a diagnosis.

When the organisation has no visible weakness. Forcing findings to have something to say is the fastest way to look ridiculous. If everything is clean, say so and move on.

When the contact has no power. A brilliant diagnosis sent to someone who cannot decide anything is wasted work. Aim at the level that holds the budget.

When I would not want the assignment. A diagnosis creates an expectation. Producing one for a project I would refuse is dishonest and it will come back on me.

The internal version

One last use, less obvious.

The same exercise, done inside an organisation you already work for, is the best way to widen an assignment.

You were brought in for one module. You produce, on your own time, a short assessment of the wider estate. You hand it over with no demand attached.

That document circulates. It reaches people you would never have met. And it positions you as someone who thinks about the whole picture, not just his own task.

> **RULE**
>
> Once a quarter, in every organisation you work with, produce one page nobody asked for.
>
> A risk observed, an unmeasured cost, a dependency nobody had noticed.
>
> That page costs you an hour. It is the cheapest renewal insurance there is.

The next chapter deals with the part that follows: turning that interest into a contract, and knowing how to refuse when you should.

◆

CHAPTER 20

Closing, contracting, refusing

> The clauses that protect you, the ones that trap you, and the four signals that should make you walk away from a contract you could have signed.

Winning an assignment is not the hard part. The hard part is winning the right one, on the right terms.

I have signed contracts I should have refused. Each time, the warning signs were there from the first meeting. I read them and I ignored them, because the revenue was tempting.

The most expensive contract is not the one you lose. It is the one you win badly.

Closing

A few things that work, with no technique or manipulation.

Sum up their problem before talking about your solution. If your summary is accurate, they will confirm it. That confirmation is the real agreement. Everything after it is detail.

Give a range, not a number. A single price invites negotiation on that number. A range with the criteria that move it inside the range moves the discussion onto scope, which is where it belongs.

Propose a first bounded step. A short diagnostic phase, a pilot on one module, a prototype. That reduces the perceived risk on their side and lets you assess the reality of the situation on yours.

Say what you will not do. Explicitly. It reassures far more than a list of everything you can do.

> **THE MOST USEFUL QUESTION**
>
> At the end of the meeting: what has to be true, in six months, for you to say this was a good decision?
>
> The answer tells you three things at once: the real success criteria, who else has to be convinced, and whether the person in front of you actually decides.
>
> If the answer is vague, the project is not mature. If it depends on someone not in the room, you are not talking to the right person.

The clauses that protect you

Written into every contract, without exception.

Scope, and the change procedure. What is included, what is not, and how a change is validated and priced. Without that clause, the scope grows silently and you finance it.

Deliverables with acceptance criteria. What is handed over, in what form, and what counts as acceptance. Silence for a defined period is acceptance.

Payment milestones. Tied to deliverables, not to elapsed time.

Intellectual property, precisely. What transfers on final payment, and what remains yours. Your reusable components, your generic tooling and your methods stay yours, with a licence gran-

ted for use. Without that clause you rebuild the same thing from scratch on every assignment.

Reversibility. What you hand over at the end: code, documentation, access, credentials, deployment procedure. Conditional on payment being up to date. Write it at the start, when the relationship is good, not at the end when it may not be.

Liability cap. Usually a multiple of the contract value. That is the clause your insurer will want to see.

Termination, in both directions. Notice period, what is due, what is handed back.

> RULE
>
> Never start work before signature, whatever the urgency.
>
> Urgency is the oldest lever there is. Anyone who genuinely needs you in a hurry can sign in a hurry.
>
> Every time I have started on a verbal agreement, either the terms shifted or the invoice was argued over.

The clauses that trap you

Now the ones to strike out.

Unlimited liability. Never. It is a bet on your whole company against one contract.

Exclusivity with no compensation. If someone wants to stop you working for their competitors, that has a price.

Unlimited maintenance included. Support has to be scoped: hours, response time, duration. Otherwise you are supplying a lifetime service for a one off price.

Full transfer of all intellectual property including pre existing work. A badly drafted clause can transfer tooling you built before the assignment. Read it.

Penalties with no symmetry. Late delivery penalties are normal. They must be matched by obligations on the client side: access provided, decisions made, environments available. Most delays come from the client, and the contract must acknowledge that.

Payment on final acceptance only. An acceptance that never comes is an invoice that is never paid.

> **THE MANAGEMENT ACCOUNTS TRAP**
>
> I was once told, on an assignment with an international client, that payment would depend on how the accounts looked at the year end.
>
> That is not a payment term. That is asking a supplier to carry a shareholder's risk.
>
> The correct answer is calm and immediate: my invoice is not indexed to your results, exactly as your customers' invoices are not indexed to mine.
>
> If that answer breaks the relationship, the relationship was already broken.

The four signals to refuse

Here are the four I have learnt to read, at a cost.

One. Time is not respected before signature. Sessions moved, late arrivals with no apology, meetings cut short. Someone who

does not respect your time when they are trying to win you over will respect it far less once they have.

Two. The decision maker never appears. You talk to an intermediary who reports upwards. Every point has to be re explained. The project will drift and you will never have a real conversation.

Three. Payment is conditional on something outside your control. Fundraising, a future result, someone else's approval. That is not a contract, that is an option they hold over your work.

Four. The previous supplier is described as incompetent, with no detail. Sometimes it is true. Often it means the organisation cannot express its needs and blames the person who tried. You will be the next incompetent one.

When four suppliers in a row have failed, the problem is very rarely the suppliers.

How you refuse

Refusing badly is expensive. Refusing well costs nothing and often earns something.

Three principles.

Be quick. A refusal after three weeks of silence is worse than an immediate one.

Be factual, never personal. Not: your project is badly framed. But: this assignment needs a level of on site availability I cannot provide.

Offer something. A contact, a piece of advice, a document. That turns a refusal into a service.

I have had prospects I refused come back a year later with a better framed project. And others who recommended me to someone else after I had turned them down.

> **RULE**
>
> Keep a written note of every refusal and its reason.
>
> After twenty of them, patterns appear. You discover which type of client, which type of project and which type of term systematically go wrong for you.
>
> That list is worth more than any commercial training.

Refusing when you need the money

The honest question, because everything above is easy when the order book is full.

When it is not, three practical answers.

Cut the scope instead of the price. Never lower your rate. Reduce what is included. Your rate is a signal, and once lowered it never rises again with that client.

Ask for a larger deposit. If the risk is higher, the security must be too. Some accept, and that tells you a great deal.

Set a defined exit. A short first phase, with an explicit review point. If it goes badly, you stop having lost weeks, not months.

And if none of those is possible, remember the arithmetic: a bad assignment does not only cost the time you spend on it. It costs the good assignment you could not take, the energy you did not have, and sometimes the reputation you cannot get back.

Nothing is more expensive than a client you should have refused.

The next part goes into the craft itself: what happens once the contract is signed, when you have to open a system nobody wants to touch.

◆

PART FIVE

The craft

What happens once the contract is signed. Taking over an orphaned system, letting artificial intelligence into a confidential estate without a single identifiable element leaving it, modernising without breaking anything, surviving your own speed, and leaving cleanly. Five chapters of pure practice.

CHAPTER 21

Taking over a system nobody wants to touch

> The first ninety days on an orphaned system. What you do, in what order, and above all what you do not touch.

A client hands you a system. The original developer left years ago. There is no documentation. There are no tests. The organisation runs on it.

Here is what you do, in order.

The first job is not to improve anything. It is to make sure the client sleeps at night.

Week one: break nothing

The absolute rule of the first days: you observe, you do not change.

The temptation is strong. You immediately see absurd things, duplicated code, dangerous practices. You know how to fix them. Resist.

You do not yet know the reasons. In an old system, the apparent absurdity is often a bug fix that became permanent. You remove it, something breaks three weeks later, somewhere with no obvious connection, and you have no way of linking the two.

What you do instead.

Take inventory. Where the code is, where the servers are, where the databases are, where the backups are, who has access. That map almost never exists and it is already a deliverable.

Check the backups. Not that they exist, that they can be restored. A backup that has never been restored is not a backup, it is a belief. Do a real restore into a separate environment. That is often the moment you discover the first serious problem.

Put monitoring in place. Services, disk space, scheduled jobs, errors. You cannot improve what you do not measure.

Listen. Users know things nobody wrote down. They know the workarounds, the risky days, the functions that have not worked for years and that everyone avoids.

> RULE
>
> The first deliverable of a takeover is not an improvement. It is an inventory and a verified backup chain.
>
> That sounds modest. It is what separates a professional from someone about to accelerate the disaster.

Weeks two to four: understand

This is the archaeology phase, and this is where modern tooling changes everything.

Extract the code. Sometimes trivial, sometimes not. On proprietary environments you have to find the export mechanism. It nearly always exists, often buried in a menu.

Split it up. The body of code is too large to process in one go. It has to be segmented into coherent units, respecting the real boundaries of the code.

Document unit by unit. For each program: what it receives, what it produces, which rules it applies, which special cases are hard coded, what it depends on.

The instruction given to the model must be strict: describe, never translate, never suggest improvements. You want a mirror, not an opinion.

Assemble. The individual descriptions become a map: dependency graph, list of business rules, inventory of customer specific behaviour, probable dead zones.

> **THE ORDER OF MAGNITUDE**
>
> Done by hand, this work occupies a team for months. That is exactly why it is never done: nobody funds months of documentation with no new features.
>
> With a properly built chain, it takes days.
>
> That does not make the work easy. It makes it fundable. That is a very different thing, and it is what opens the market.

Month two: stabilise

Once you understand, you fix what is urgent and safe.

In order of priority:

What is exposed. Interfaces reachable without authentication, remote access left open for convenience, default credentials, versions with publicly documented vulnerabilities. You start by closing the doors.

What threatens continuity. Full disks, expired certificates, scheduled jobs failing silently, logs that are not rotated.

What depends on one person. Any operation only one person knows how to perform must be documented and then automated. That is the organisation's real risk.

What depends on one workstation. There is very often a machine under a desk running a critical job. You find it, you document it, you move it.

> **WHAT YOU DO NOT TOUCH**
>
> **The business core.** Until you have tests, you do not change business logic. Ever.
>
> **Exchange formats.** Other systems depend on them, often at third parties nobody has told you about.
>
> **Naming.** Renaming for clarity looks harmless. In a system where references live in scripts, scheduled jobs and scattered configuration files, it is a high risk operation.
>
> **Anything nobody understands.** If you do not know what a module is for, do not delete it. Isolate it, watch it, wait.

Month three: take back control

At this stage you can start improving structurally.

Put the code under version control. Many old systems are not, or use a non standard proprietary mechanism. It is thankless and fundamental work.

Rebuild environments. An isolated test environment, with representative anonymised data. Without it, no change is safe.

Write the first tests. Not everywhere. On the critical paths, the ones that make money or create legal exposure. Those tests are

your safety net for everything that follows.

Automate deployment. Even simply. A manual deployment is a permanent source of error and a point of human dependency.

The human dimension

An aspect technique does not cover, and which decides everything.

There is nearly always somebody inside who knows part of the system. That person is in a delicate position. Your arrival may look, to them, like a judgement that they were not enough.

How you handle that.

Praise them, explicitly. This system held together because of them, in difficult conditions, without resources. It is true, and it must be said to the director, in writing.

Document with them, not instead of them. They validate, they correct, they add. They become a co owner of the documentation.

Never take a criticism over their head. Any finding touching their area is shown to them before it is shown to their management.

The system is technical. The assignment is human. A takeover almost always fails for human reasons.

The deliverable that changes the relationship

At the end of those three months you hand over a document that did not exist.

It contains the full inventory, the real architecture, the extracted business rules, the dependencies, the ranked risks, the operating

procedures, the state of the backups, and the modernisation plan.

That document has a precise effect: it turns an opaque asset into an understandable one. For a director, it is the first time he genuinely owns his information system.

> **RULE**
>
> Invoice understanding before you invoice transformation.
>
> It is easier to sell, less risky to execute, and it builds the trust the rest of the work needs.
>
> And if the client stops there, he will have received something genuinely useful. That is what brings him back.

The next chapter covers the problem that blocks all of these assignments as soon as you want to use artificial intelligence: how to analyse confidential code without ever letting it leave.

◆

CHAPTER 22

The airlock

> Letting artificial intelligence into a confidential system without a single identifiable element leaving it. The full mechanism, and the trade off you have to settle.

This is the lock that blocks assisted modernisation in every regulated sector. And it is, in my view, the most interesting technical subject of the decade in this niche.

The problem

You want a language model to help you understand and convert business code. That code contains:

- real data hard coded into the processing, including personal data;
- table, schema and field names that describe the company's activity;
- business rules that constitute the trade secret;
- sometimes technical credentials and infrastructure paths.

A financial institution, an insurer or a logistics operator will refuse to let that out. Including with a contract guaranteeing no retention and no training.

That refusal is rational. The risk is not only contractual, it is regulatory and competitive.

> **You are not selling artificial intelligence. You are selling the right to use it without losing control.**

What is actually sensitive

First distinction, and it is poorly understood.

Personal data. In old systems it is generally not in the source code but in the files the code manipulates. The code describes the structure, the files hold the material.

Be careful though: there are cases where real data sets are written directly into the sources. That happens more often than people think, and it is the most dangerous case.

Legal status: unambiguous. It is personal data, it must be masked before anything leaves.

Business identifiers. Names of variables, tables, fields, schemas. These are not personal data. They are intellectual property and trade secrets.

Legal status: more nuanced. Many organisations send their code as it is, protected by a contract. But those holding old systems, which is exactly our client base, often refuse whatever the guarantee.

The business logic itself. The chain of rules is the heart of the competitive advantage. It is what worries directors most.

The architecture of the airlock

Three stages.

Stage one: mask, locally.

A process runs entirely on the client's infrastructure. It detects sensitive elements and replaces them with neutral tokens. The mapping table between the original and the token is kept locally and never leaves.

Technical points to watch:

- The replacement must be consistent across the whole corpus. The same identifier always gets the same token, otherwise the model loses track of the dependencies.
- The replacement must preserve the physical structure. In old languages, the position and the length of fields carry meaning. Replacing a field with a token of a different length breaks the semantics.
- The substitute data must stay realistic. Replacing a date with an arbitrary string produces code that no longer makes sense.

Stage two: process, outside.

The masked corpus is sent to the service. Nothing that comes out of it allows a person to be identified or a business name to be reconstructed.

Stage three: unmask, locally.

The result comes back. The mapping table is applied in reverse. The produced code recovers its real names.

The central trade off

Here is the most interesting point, and the one nobody anticipates.

Masking business identifiers degrades the quality of the result.

A model reading a clearly named field understands that it is a personal identifier. It infers validation rules, format constraints, ap-

propriate handling.

The same field renamed to a neutral token tells it nothing. It loses the semantic context, so it loses relevance.

> **THE QUESTION TO SETTLE EXPERIMENTALLY**
>
> There is no theoretical answer. You have to measure.
>
> The method: take a representative program, produce a conversion with no masking to establish a baseline, then produce the same conversion with identifiers masked, and compare.
>
> You measure functional fidelity, preservation of business rules, readability of the result.
>
> Then you look for the balance point: which elements absolutely must be masked, which can stay, and how much quality you accept losing for how much confidentiality gained.

There is a middle route that works well: mask while preserving the semantic category. A national identifier does not become an anonymous token, it becomes a typed token that indicates the nature of the field without revealing the company's real naming.

The model keeps the useful information. The organisation does not reveal its nomenclature.

The hybrid approach

Second pillar, complementing the airlock.

Language models hallucinate massively on old and proprietary languages. They invent instructions that do not exist, with total

confidence. On the least publicly documented technologies, the effect is systematic.

The answer is to stop asking them for what they cannot do.

What you give to structural analysis, locally:

- Parsing the code into a tree representation.
- Extracting dependencies between units.
- Identifying data structures and their overlays.
- Detecting conditional branches per customer.
- Detecting dead code.

Those operations are deterministic. They cannot hallucinate. They produce facts.

What you give to the model:

- A natural language description of what a block does, once its structure has already been extracted.
- A proposed modern equivalent, built from a structured specification rather than from raw code.
- Writing the documentation.

What you check afterwards:

- Structural comparison between the original and the result.
- Verification that every identified business rule is present.
- Running tests built from the extracted cases.

You do not send the code to the model. You send a structured description of the code, produced locally by tools that do not get it wrong.

That pairing of deterministic symbolic processing with neural processing is what is called a neuro symbolic approach. The name is grand, the idea is simple: each side does what it is good at.

The hidden benefit

There is a major side effect, and it is economic.

Sending a million lines to a model is expensive. Sending a structured description of the fifty relevant lines costs a tiny fraction.

Filtering up front cuts the cost by a large factor. On a full takeover project, that is the difference between an acceptable budget and a prohibitive one.

> **WHAT THE FULL CHAIN DELIVERS**
>
> **Confidentiality.** Nothing identifiable leaves the client's infrastructure.
>
> **Reliability.** Structural facts come from deterministic analysis, not probabilistic generation.
>
> **Cost.** An order of magnitude reduction in consumption.
>
> **Defensibility.** The process can be documented and audited, which matters as much as the result.

The commercial argument

Let us finish with what turns this architecture into a market advantage.

Many organisations already use assistance tools on their code, with contracts that protect them. They do not need an airlock.

But our client base is not that one. It is the organisations holding old and critical systems, who refuse categorically, whatever the contract says.

For them the argument is decisive: you do not have to trust a model provider. Nothing identifiable leaves your systems, because the masking happens at your end, first.

> **RULE**
>
> In a market where everyone is selling artificial intelligence, the rare position is not to sell more of it.
>
> It is to sell the guarantee that it can be used without putting yourself in danger.

The next chapter describes modernisation itself, once understanding is in place and the airlock is working.

◆

CHAPTER 23

Modernising without breaking anything

> The strangler pattern, the database as a meeting point, pulling out business rules. How you transform a system without ever stopping it.

You understand the system. It is stable. The client trusts you. Now you modernise.

Here is the method, with its variants and its traps.

The guiding principle

Every step must satisfy three conditions:

Reversible. You can go back within minutes.

Verifiable. You can prove that behaviour has not changed.

Interruptible. If the budget stops, what has been done stays useful and coherent.

A modernisation that does not satisfy those three conditions is a rewrite in disguise.

Sell reversibility. That is what the director is buying, not modernity.

The strangler pattern

This is the backbone of the method. The name comes from a botanical image: a plant that grows around a tree, gradually wraps it, and ends up occupying its place once the tree has gone.

Step 1. Put the facade in place.

You put an interface layer in front of the existing system. All new calls go through it. At the start it does nothing but forward to the existing system.

That step changes nothing functionally, and that is precisely the point. It is risk free, it is quick, and it creates the control point that makes everything else possible.

Step 2. Redirect, function by function.

You choose a peripheral function, low risk, well understood. You implement it in a modern module. You configure the facade to route it to the new module.

If something goes wrong, you switch back by configuration. In seconds.

Step 3. Repeat, in the right order.

From the periphery towards the centre. Read before write. Rare functions before critical ones.

Step 4. Remove what is no longer called.

When a function of the old system has received no call for a long enough period, you can consider removing it. With monitoring in place first, never on static analysis alone.

> **THE CLASSIC MISTAKE**
>
> Starting with the central module because it is the most important.
>
> That is the one most often called, most coupled, least understood. A failure there stops the whole modernisation programme and destroys trust.
>
> You start with the dull and peripheral. You build a run of successes. The accumulated trust funds the move to the hard parts.

The database as a meeting point

A very effective variant, and academically inelegant, which makes it all the more interesting.

Rather than making the old and the new talk through application interfaces, you have them share the same database.

The new module reads and writes the same tables as the old one. Both worlds share the state.

Advantages:

- No data migration, so no high risk switch over window.
- Immediate coexistence.
- Trivial rollback.
- The new module can be built and tested in parallel, without heavy coordination.

Drawbacks:

- The old data schema contaminates the new code.
- Transactional consistency between the two worlds needs attention.

- You inherit the modelling constraints you specifically wanted to escape.

> **WHEN TO USE IT**
>
> This approach suits the case where the database is healthy and the code is sick. That is the most common case: the data models of these systems were often well designed originally.
>
> It is not advisable when the data model itself is the problem.
>
> When in doubt, start there. It is faster, less risky, and you can always clean up the schema later.

Pulling out the business rules

This is the transformation with the biggest effect on the total cost of ownership.

A reminder of the problem: the logic specific to each customer is hard coded, as stacked conditional branches.

The transformation happens in four stages.

One. Inventory. Every conditional branch tied to a customer, a region, a contract. The map from chapter 21 has already located them.

Two. Classify. Many of those branches actually express the same thing with different values. You identify the families.

Three. Externalise. Each family becomes a configuration parameter. The branch disappears from the code, the value moves into a per customer configuration table.

Four. Verify. For each customer you replay a representative set of cases and compare the results before and after. Bit for bit if

necessary.

The result: adding a customer no longer requires changing the code. That is a change of economic nature.

> **WHAT THAT CHANGES FOR THE CLIENT**
>
> Before: onboarding a new customer requires a code change, a development cycle, a regression test across every other customer, and a release.
>
> After: it requires one line of configuration.
>
> The marginal cost of acquiring a customer collapses. That is an argument a director understands immediately, unlike anything about code quality.

Tests, the precondition for everything

None of the above works without a safety net.

The aim is not to cover the whole system. That is unrealistic and it is not necessary.

The aim is to cover what makes money and what creates legal exposure. On a business management system, that is usually a small part of the code.

The most effective technique on legacy is the characterisation test: you do not try to describe the correct behaviour, you record the current behaviour and check that it does not change.

That is philosophically debatable, since you also freeze the bugs. It is operationally excellent, because in an old system the bugs are part of the implicit contract with users, who have built their own workarounds around them.

In an old system, fixing a bug can break a business process that organised itself around it.

The cloud question

A word on cloud migration, because it is consistently framed wrongly.

The question is not on premises or cloud. The question is: what has to move, and why.

Three configurations, each legitimate.

Everything on premises. Justified when sovereignty is a hard constraint, when the load is stable and predictable, when the existing kit is written off. Many organisations would pay more in the cloud.

Everything in the cloud. Justified when the load is variable, when you need elasticity, when you want to remove hardware management.

Hybrid. The most common configuration in practice. Sensitive data stays on premises, elastic workloads and artificial intelligence services sit in the cloud. Mechanisms exist to attach local servers to a remote control plane, which gives you the modern tooling without moving the data.

> **RULE**
>
> Never migrate in order to modernise. Migrate to solve a precise, named problem.
>
> An old application moved unchanged into the cloud is still an old application, now with a monthly invoice and a new dependency.

The full order

Let us recap the sequence.

1. Understand. Document, map, extract the rules.
2. Stabilise. Close the doors, monitor, back up, reduce human dependency.
3. Equip. Version control, environments, tests on critical paths, automated deployment.
4. Put the facade in place.
5. Redirect gradually, from the periphery towards the centre.
6. Externalise the business rules.
7. Decide on infrastructure, driven by a real problem.
8. Remove what is no longer used.

Each step produces value on its own. The client can stop at any of them and keep the benefit of what has been done.

It is the only honest way to sell a modernisation: every step justifies itself alone.

The next chapter deals with a problem no technical method solves: what happens when you deliver far faster than the market expects.

CHAPTER 24

Delivering too fast

> Several weeks sold, two days used. The problem is not technical, it is economic. And it is handled before delivery, not after.

Let us go back to the prologue, but this time from the consequences side.

One precaution before going in. What follows targets no company in particular, and certainly not those I have worked for. I have observed exactly the same phenomenon in almost every services company I have come across since, here and in Europe, at my clients as much as at their suppliers. The problem is not the mindset of the teams or the directors, who are usually acting in good faith. The problem is arithmetic, and it comes entirely from one thing: billing by time spent.

You deliver in two days what was sold as several weeks. You think you have performed. You have created an accounting problem.

A business model indexed on time spent cannot, structurally, greet a productivity gain as good news.

Why it is a problem

Three reasons, and they compound.

The first is economic. Time and materials sells days. Dividing the time needed by ten divides the invoice by ten, at equal scope. It is not a question of willingness: no structure can absorb that without changing its unit of sale.

The second is social. Your colleagues do the same work at the normal pace, which is the reasonable pace of the trade. If your pace becomes the reference, an unfair standard settles in for everyone. That is good for nobody, yourself included.

The third is structural. Part of the roles in any organisation exist because certain problems are difficult. Making a problem easy moves those roles, and moving them takes time and preparation.

That is not malice. It is economic inertia, and it is perfectly rational from the point of view of those who live with it.

THE TEST I SET

I regularly put this case to engineers, in training or in interviews. It is a teaching case, not a memory.

You document a million lines in two days when several weeks had been sold. Four options: say nothing and fill the calendar, tell your management, tell the client, tell both.

The answers spread across all four, and each person defends theirs with good arguments.

Those who choose to tell management talk about recognition and a possible pay review. In practice, what most often happens is a wider scope with no change in the unit of billing.

Those who choose silence get free time and learn nothing.

There is no right answer as long as the unit sold is time. That is precisely what the case is meant to show: the problem is in the contract, not in the people.

What I do now

In my own structure the problem disappears, because I do not sell time.

I sell an outcome. The price is set on the value of the problem solved and the risk carried, not on hours used. Whether the work takes a day or three weeks does not change the price.

That is the only model compatible with current productivity. And it is the underlying reason I set up a company: not for the status, for the unit of billing.

> **RULE**
>
> As long as you bill time, your productivity works against you.
>
> The day you bill an outcome, it works for you.
>
> Same technical act. Two opposite economies.

The effect on the client

That leaves a delicate question: what do you tell the client?

My position is transparency about the method, never about the duration.

I do not say how long it took me. That is not his business, exactly as a craftsman does not itemise the duration of each movement.

What I do explain is the method. I fully own the use of intelligent assistance. I even explain it in detail, because it is part of what the client is buying: the certainty that the work is done with the best tools available, and not by hand out of nostalgia.

Nobody is buying my hours. They are buying the fact that the problem goes away.

Be aware that some clients react badly. I have seen two typical reactions.

The first. If it is that quick, why that price? Answer: you are not paying for the duration, you are paying for the fact that it is possible at all. The price reflects years of experience that let me know what to ask for, in what order, and how to check it.

The second, more interesting. If it is possible here, why does everything else take months with my other suppliers?

That question is explosive inside the client's ecosystem, and you have no interest in answering it head on. I answer factually: organisations have different constraints, approval processes and decision cycles. That is not untrue, and it avoids turning you into an accuser of other suppliers, a position that never pays.

Speed as a commercial argument

Used well, that speed becomes a decisive advantage.

It lets you give. A deliverable that costs a few hours can be handed over to open a door. That is the whole of chapter 16.

It lets you take risks. You can accept a fixed price commitment on a scope a competitor would refuse, because your real production cost is far lower than his.

It opens a market that was out of reach. Documenting an old system used to cost months, so nobody funded it, so nobody sold it. At a few days it becomes an ordinary service. You are not taking market share from a competitor, you are creating a market.

THE FIXED PRICE CALCULATION

A competitor billing by time on a system takeover will estimate several person months. His price will be high, and his client will hesitate.

You can propose a much lower fixed price, with a comfortable margin, because your real cost is of a different order.

The client compares two proposals. Yours is cheaper and commits to an outcome. You win, and you win with a better margin than if you had billed by time.

It is the only situation where lowering the price increases profitability.

The trap of speed

A serious warning, because I have seen this trap close.

Speed creates an expectation that becomes permanent.

A client used to receiving a change in two days considers two days to be the norm. The day something takes two weeks for good reasons, he does not understand. You have become slow.

Two protections.

Do not deliver everything the moment it is ready. That is not dishonesty, it is expectation management. The time between finishing the work and delivering it is used for review, testing and improvement. It also produces higher quality.

Communicate about milestones, not about execution. You announce delivery dates. You do not narrate the internal process.

THE EXHAUSTION TRAP

There is a more serious and more personal trap.

When everything becomes quickly possible, the temptation is to do everything. Several projects in parallel, evenings that stretch out, the heady feeling of unlimited capacity.

It is not unlimited. I wore myself out twice, and both times it was self inflicted, out of ambition, with nobody asking me to.

The tool removed the technical limit. It did not remove the human one. You remain responsible for every line delivered, and that responsibility cannot be automated.

Set yourself limits that nothing forces you to respect. That is exactly why they are necessary.

What to take away

Radical productivity is an asset, provided you are in the right economic structure.

In a model that sells time, it is an accounting difficulty to be handled carefully, and nobody can do much about it until the unit of sale changes.

In a model that sells outcomes, it is a decisive competitive advantage.

The choice between the two is not ideological and says nothing about the worth of the people working in either. It simply decides where the gain you produce ends up.

The question is never whether you can do it fast. It is who pockets the difference.

The next chapter closes the cycle of an assignment: how to leave cleanly, and why you must never create dependency.

◆

CHAPTER 25

Leaving cleanly

> Handing over the keys, documenting everything, holding nothing back. Why refusing to create dependency is the best long term commercial strategy.

An assignment ends. That is the moment most suppliers handle badly, and it is the one that decides whether you come back.

The temptation of dependency

The classic model of many suppliers rests on a simple idea: the client must not be able to do without you.

That gets built in several ways, often without conscious intent:

- minimal documentation, the knowledge stays in people's heads;
- access retained, the client does not hold his own keys;
- needlessly complex architecture, unreadable to an outsider;
- proprietary or bespoke tooling only the supplier knows;
- undocumented manual deployment.

It works, in the short term. The client is captive, he renews, he does not dare leave.

It fails, in the medium term, for a simple reason: the client knows he is captive. He resents it. And the day he finds a way out, he takes it and never comes back.

Captivity produces recurring revenue and resentment. Resentment always wins in the end.

My position

I do the opposite, systematically and openly.

Everything is handed over. Code, documentation, access, procedures, accounts, domain names. The client owns it all.

That is not generosity. It is a calculation, and it rests on three observations.

One. A client who knows he can leave is no longer afraid to engage you. Fear of lock-in is the first obstacle to signing, especially for someone who has been locked in once already.

Two. A client who owns everything and comes back anyway is a client won by value, not by constraint. That relationship is stable.

Three. A client who leaves on good terms recommends you. A captive client who escapes tells a different story.

> RULE
>
> Always build so that somebody else could take over.
>
> It is an immediate commercial argument, it is a healthy technical discipline, and it protects you too: you can end an assignment that is going badly without guilt and without conflict.

The handover, in detail

Here is what a full handover actually includes. Each of these points has been, at least once, the source of a problem.

The code. Transfer of ownership of the repository, not just access. The difference is essential: access can be revoked, ownership is kept.

The secrets. Interface keys, certificates, service credentials. They must be passed through a secure channel, then rotated by the client. A secret that has been passed on is a compromised secret.

The infrastructure. Provider accounts, configuration, associated payment methods. This is often the most painful point, because those accounts are frequently opened in the supplier's name.

The domain names. Transfer of ownership, not just technical management. A domain left with the supplier is a potential hostage, even unintentionally.

The deployment chain. Configuration, triggers, variables. The client must be able to release without you.

The operating documentation. How to start, stop, back up, restore, diagnose. That document is worth more than the code.

A handover session. One or two hours, screen shared, going through everything and having the client perform the operations himself.

WHAT ACTUALLY TAKES THE TIME

The technical part of the transfer takes minutes.

What takes the time: multi factor authentication, propagation delays, certificate validations, ownership changes requiring email confirmation on an address whose password nobody has any more.

Allow two hours for the session and expect follow ups. Do it at a time when you are available, never in the rush of a contentious contract ending.

The contentious exit

You also have to prepare for the bad scenario.

When a relationship deteriorates, the exit must be cleaner still, not less clean.

My method:

One. Everything in writing. From the moment the relationship tightens, nothing by phone without written confirmation afterwards.

Two. No withholding. Even in a payment dispute, you do not cut off a service in production. You apply the contractual clauses, you do not take hostages. Leverage obtained that way costs more than it earns.

Three. No public argument. Not on social platforms, not with other clients. The market is small and it has a memory.

Four. You leave everything. Including when the other side has not honoured its part. What was produced is handed over. That is

the position that leaves you beyond reproach, and it has real strategic value.

In a conflict, the one with the least to lose is in the stronger position. Never create a situation where you have something to lose.

What you keep

A clarification, because this is not about giving everything away.

What you hand over: everything relating to the client. His code, his data, his access, his documentation.

What you keep: your own tools, your own methods, your own reusable components developed beforehand and used across several assignments.

That distinction must appear in the contract, from the start. A simple clause: the deliverable belongs to the client, pre existing generic components remain the property of the supplier and are granted under a licence to use.

Without that clause, you end up giving away your tooling on every assignment.

The right to refer to the work

A frequent question, and often badly negotiated.

Can you mention an assignment? That depends on what was agreed, and it must be agreed explicitly.

My practice: I systematically ask for the right to mention the sector, the nature of the work and the technologies, without naming the client. That is almost always granted, because it costs nothing.

The right to name is negotiated separately and is rarely obtained at the start. It is obtained afterwards, when the assignment has gone well, by simply asking.

> **RULE**
>
> Ask for the recommendation at the moment of success, not at the moment of departure.
>
> The best time is just after a successful delivery, while the satisfaction is fresh. Three months later the effect is forgotten and the request looks self interested.

Reversibility as a product

Let us finish with a reversal.

Reversibility can be sold in its own right.

Many organisations are locked in with a supplier and know it. They cannot leave because they own neither their code, nor their documentation, nor their access.

A control recovery assignment is a service in itself: recovering the assets, rebuilding the documentation, taking back the infrastructure, reducing the dependency.

It is a short assignment, with high perceived value, and it nearly always opens onto a lasting relationship. A client who has just discovered what it means to own his own system will not go back to the one who was holding him.

Freeing a client from another supplier is the best commercial introduction there is.

That is the craft. The next part, the last one, deals with what remains once the assignments pass: what you pass on, what you build around you, and what you would do differently.

◆

PART SIX

Passing it on

What remains once the assignments are over. What around thirty interns taught me about supervision, what can be built around you without leaving your country, and an honest account of what I would do differently.

CHAPTER 26

Mentoring

> Around thirty interns in three years, across three continents. What that taught me about the real level, about what unlocks a young engineer, and about the mistakes I made.

I have supervised around thirty interns. Twelve in a banking context, about twenty in a services company, then an international team in my own structure.

I will describe what I took from it, including the parts that do not flatter me.

International recruitment

When I opened an applied research programme for my company, I received more than a hundred and ten applications, from everywhere: United Kingdom, France, Poland, Belgium, Romania, United States, Tunisia, Lithuania, Gulf countries, Syria, India, Algeria.

The panel was impressive on paper. Doctoral students, engineers from large companies, very well presented profiles.

What I found on examination was brutal.

A majority of identical applications. A stack of libraries, wrappers around language models, connectors between services. Everyone is doing artificial intelligence, very few understand the engineering underneath. Using a wrapper is not engineering.

Career changes with no foundation. Paths that went through short courses, with no fundamentals, and a very polished presentation.

Inflated experience. I came across several applications mentioning the same organisation showing hundreds of active interns, a practice that artificially inflates a headcount on display.

Candidates reading their answers. In interview, out loud, from a screen, with the characteristic lag.

Personal branding has massively replaced substance. That is an opportunity for those who kept the substance.

Who actually performed

It was neither the doctoral students nor the profiles with prestigious logos.

It was fourth year engineering students, two of them from Algerian schools, plus one candidate from the Middle East.

On fundamentals, depth of reasoning and the ability to take apart an unfamiliar problem, the gap was clear.

One story stayed with me. The candidate from the Middle East joined his interview powered by a solar panel, in a country where electricity is intermittent. He was on time, precise, and he had already converted old code to a modern technology by hand.

On the other side, I had had candidates from comfortable countries who did not turn up to appointments they had set themselves.

> **RULE**
>
> The most predictive factor is neither the degree nor the experience on display. It is the ability to function in degraded conditions.
>
> Someone who has learnt to produce without resources will produce with them. The reverse is not true.

What actually unlocks a young engineer

Three things, and none of them is technical.

The first: take the pressure of the result off them.

A student I was supervising wrote me a message I still keep in mind. She described an unproductive productivity: the constant pressure to finish quickly, which made her skip steps, go straight to the code, and never let her reasoning unfold. She said she was becoming like everyone else, without letting her engineering skills show.

That is a diagnosis of rare clarity, and it describes exactly what the system produces.

My answer was the opposite of what people usually hear: take your time, nobody is chasing you, and if you take three times as long because you understood it deeply, you will have won.

The change of pace was immediate, and the quality of the work followed.

The second: give them access to professional tools.

This is massively underrated. A student with the same tools as an experienced professional changes category within weeks. That access costs almost nothing when you have the programmes described in chapter 14.

And it produces a side effect: mastery of those tools is itself a differentiator on the market.

The third: show them the real landscape.

A student only knows school. He does not know how an organisation works, what a production system is, how a client decides, what the career paths he is being sold are actually worth.

Every time I have shared an observation from the field, even an unpleasant one, the effect was the same: the person stopped comparing themselves to an imaginary standard.

Impostor syndrome is not cured by encouragement.
It is cured by information.

The method that works

What I apply, after several iterations.

No binding schedule at the start. Objectives are not fixed before the exploratory phase ends, because you do not yet know what you are looking for. That makes the initial phase uncomfortable, and it has to be announced.

Detailed objectives as soon as the direction is clear. Constantly changing objectives is the number one cause of a failed internship.

No obligation to turn the camera on. You focus on the content.

The right to stop, stated from the start. Studies come first, always. Someone who knows they can leave without conflict stays longer and works better.

No formal interim report. Working sessions with screen sharing, where you see what is really blocking.

Systematic recognition on the way out. Certificate, written recommendation, public mention of the work, a bonus when it is deserved.

> **THE DETAIL THAT COUNTS**
>
> On end of internship documents, I write engineer rather than intern when the level justifies it.
>
> That is not indulgence. A document that expires in the reader's perception after six months is worth nothing. A document describing the real level of the work done stays useful for years.
>
> The vocabulary chosen on a certificate concretely changes someone's trajectory. It is free and it is decisive.

My mistakes

I have to name them.

First mistake: believing that the quality of the supervision was enough.

I supervised around thirty people on my own time, with real investment. The direct return was lower than I imagined, and that is normal: each of them followed their own path, and nothing obliged them to come back.

The mistake was mine. I was implicitly expecting something I had never stated.

There is no ingratitude on their side. It is mechanical: what is given without a frame simply becomes the norm, and a norm is no longer negotiated.

Second mistake: not formalising what each side expected.

An internship with nothing written about what each party expects produces misunderstandings on both sides.

Third mistake: underestimating the cost to me.

Two episodes of burnout. Nobody imposed them on me: I was adding supervision to an already full load, on my own initiative, without removing anything elsewhere and without saying a word about it.

> **TRAP**
>
> Good supervision, in many organisations, rests entirely on the personal willingness of an engineer giving up their free time, with none of it appearing in their objectives or their pay.
>
> It works as long as the person holds. Then they stop holding, and it all stops at once.
>
> If you supervise, negotiate dedicated time. Otherwise, limit the number. Two people well supervised are worth more than ten badly followed, for them and for you.

What I take from it

Despite that mixed record, I would do it again.

Because I learnt more from supervising than from being supervised. Working with students from specialisms I did not master, I gained in months what I would not have gained alone in years.

Because some people genuinely changed direction, and a few told me so in a way that is worth more than any contract.

And because it is the only mechanism that raises the collective level. The rest is a transfer of value between players who are already established.

You do not progress by keeping. You progress by lifting others, provided it does not cost you your health.

The next chapter widens that idea: what can be built around you, from home, and the limits I ran into.

◆

CHAPTER 27

Building an ecosystem from home

> Public sessions, community platforms, migrating a university system. What worked, what took longer than expected, and what I learnt about running collective projects.

For three years I carried a significant number of collective initiatives. None of them had been asked of me. All of them were run on my own time.

Here is the honest account, successes and failures.

What worked

Free public sessions.

Two technical sessions of two hours each, open to everyone, free of charge, with a certificate. One on cloud computing and operating practice, one on web development. Close to a hundred participants each time, students and professionals.

The content was built from the field, not from textbooks. The recordings were left freely available.

Why it worked: there was no free equivalent with that level of applied content. And the format was short. Two hours, people come. A three month course, people drop out.

Organising sessions with outside speakers.

A session on the legal framework for personal data protection, delivered by a senior consultant, in partnership with a student club.

Another on product management, with a speaker from industry and higher education.

That format is excellent because it does not rest on you. You organise, you provide the frame, somebody else brings the content. It is repeatable.

What I personally gained: the data protection session let me ask my own questions about concrete cases. I learnt as much as the participants.

Migrating a university system.

Several years after finishing my studies, I came back to the management platform I had designed during my final year project, which was deployed across several institutions.

It had not been updated for a long time. I jumped several major versions in one working session, with the tooling described in chapter 15. Every regression was dealt with. The visual redesign came on top.

It did not take months. It took a few hours.

> **WHAT THIS EPISODE TEACHES**
>
> A piece of software that is useful does not die because it ages. It ages because nobody has time to come back to it.
>
> What blocked that update was not technical difficulty. It was the assumed cost of the operation, which made it impossible to schedule in an already full calendar.
>
> When that cost falls to a few hours, a platform everyone thought was frozen comes back to life. That is true for an institution as much as for a company.

What took longer than expected

The community internship platform project.

The idea: a free and open platform, bringing together internship offers and requests nationally. A company database, offer posting, request posting, filtering by skill and availability.

The project was carried by a team of students as part of their end of cycle project, with professional supervision, a survey of companies and candidates, several requirements gathering sessions and a studied legal framework.

The work delivered was serious and the result was good. I want to say that clearly: the team did what was needed, and more. The project was in fact successfully defended in the academic setting.

It has not yet been launched publicly. It is on hold, with the organisation supporting it, and it can restart at any time.

The reasons for that pause are not technical. Technically, that project can be built today in a few days.

THE REAL CHALLENGES OF A PROJECT LIKE THIS

First, the material to publish. A matching platform only has value if both sides feed it. Until the offers arrive in volume, the tool stays empty, whatever its quality.

Second, the question of ratings. Allowing organisations to be rated raises delicate legal questions, and that risk must never rest on students. It is a design choice to settle very early.

Third, a difference of expectations. The student comes looking for recognition and supervision. The supervisor, often already at capacity, believes they are giving as much as they can. Both are right from their own point of view. A tool does not solve a problem of available time.

Fourth, governance. A project aimed at the community must be clear, from day one, about who owns it, who maintains it and who takes over if the lead stops. Without that, it depends entirely on the availability of a few people.

There is nothing bitter in what I take from it. The students who carried that project learnt a great deal, so did I, and several of them went on to excellent careers. That is already a result.

The lesson is about me, not about them: I had underestimated that the problem was not the absence of a tool.

Trying to lift things up without wrecking yourself is a far harder exercise than it looks.

The observation on engagement

I recently proposed a paid competition around the interface of a university platform. Prize funded by me, accessible subject, no constraints.

There were few responses.

I will not draw a judgement about anyone from that. Two explanations seem plausible to me, and the second concerns me more than the others.

The first: the calendar. A proposal landing during exams or internship hunting has no chance, whatever its quality.

The second: my proposal was probably not clear enough nor attractive enough given everything already competing for a student's attention today. I judged it with the reference points of my own era, which is rarely a good method.

Both are probably partly true.

What I take from it operationally: I redirected my energy towards people further along in their path, where the ratio between effort invested and result obtained is more favourable.

> **RULE**
>
> A collective initiative needs three things: a lead who lasts, a legal structure that protects, and a clear benefit for each participant.
>
> If one is missing, do not start. You will produce frustration and burn your credit.

What I would do differently

Formalise the frame from the start. Who owns what, who decides, who takes over if the lead stops.

Choose projects that survive my departure. A recorded session stays available. A document circulates. A platform requiring permanent maintenance dies with the person carrying it.

Reduce the ambition and increase the frequency. A two hour session every two months produces more cumulative effect than a one year project that is never finished.

Clarify governance on day one. A project aimed at the community must say from the start who owns it, who maintains it and under what licence it lives. It is a question of clarity, not of distrust.

The other route

There is a more effective way to contribute, and I discovered it late.

Rather than building collective structures, transfer resources directly to individuals.

Concretely: give access to a professional tool to a student who would never have it. Pay for half a day of knowledge transfer with a top student, which recognises them and brings you up to date. Give a domain name and a portfolio to someone looking for an internship. Write a detailed recommendation. Share a prospecting tool with someone struggling to find an opportunity.

None of that produces an announcement. None of it produces a group photograph. And the effect on that person's trajectory is incomparably greater.

Collective initiatives make noise. Individual transfers change lives.

The next chapter closes the book with an account free of flattery: what I would do again, what I would not, and what I have not yet solved.

◆

CHAPTER 28

What I would do differently

> The account free of flattery. The mistakes, the regrets, what remains unsolved, and the six decisions I would make in exactly the same way.

End of the journey. Now I owe an honest account.

What I would not do again

Sacrifice with no frame.

I gave an enormous amount, with no frame, for three years. Public sessions, supervision, community projects, initiatives nobody asked for, work in the evenings, at weekends, during holidays.

The return was far below the investment. Not zero, but far below.

What I would do instead: give as much, but always with a defined and dated return. Not necessarily financial. Defined.

Technical overcompensation.

For several years I answered relationship difficulties by producing more. Faster, more, better. I thought quality would settle the question.

It never did. It made me useful and isolated.

What I would do instead: invest in relational intelligence at the same level as in technique. That is not selling out, it is a skill, and it can be learnt.

Contempt expressed out loud.

When I implied that my classmates were following the crowd without questioning it, I was perhaps right on the analysis and clearly wrong on the delivery. Each of them had their reasons, and many did very well. That clumsiness cost me relationships, lastingly.

What I would do instead: keep the analysis, drop the judgement. You can make the opposite choice without telling anyone.

Looking for a path inside a system that does not offer one.

I spent about three years looking for how to succeed inside the existing system. I did the full tour: an institutional project deployed, a multinational bank at the core of its applications, a services company working for foreign clients, supervision, recruitment, interviews, freelancing, independent status, collective initiatives.

I eventually accepted that there was no default path.

I could have saved that time if somebody had told me there was no path. Nobody told me, because nobody says it.

What I would do instead: accept sooner that you have to draw your own route. Time spent looking for a model is time lost when the model does not exist.

Ignoring the signals.

Every time somebody did not respect my time, moved an appointment he had set himself, cut a session short, I carried on anyway. Every time, I paid for it.

What I would do instead: treat respect for time as a knockout criterion, with no exception.

What I do not regret

Never having filed an application to leave.

Not one application, not one form. I have never spent a second on that question.

That is not ideology. Looking back, and after watching dozens of trajectories closely, it was the right calculation. I kept the absence of debt, the local cost of living, the family anchor, and I built a sales channel outwards.

Having chosen the specialism nobody wanted.

That determined everything else. A cross cutting view, immediate access to the business fabric, a rare skill.

Having taken an interest in old systems.

The most counter intuitive and the most profitable choice. Where nobody goes, there is a budget, a structural need and no competition.

Having created a company rather than waiting for a promotion.

The company changed the unit of billing, so it changed the whole economy. It cost little and unlocked everything.

Having given before asking.

Despite the disappointments, that is what opened a market where I knew nobody. Mentoring given with nothing expected in return gave me access to directors no sales approach would have reached.

Having refused.

Every refusal cost me immediate revenue and spared me a lasting problem. The portfolio is smaller and infinitely healthier.

What is not solved

I am not going to finish on a triumphant conclusion. Here is what remains open.

Dependency on me personally.

Everything rests on me. I am the bottleneck of every project. Attempts at recruitment have not worked out, for various reasons: minimum wage regulations, differences of outlook, lack of rigour, or simply because I have not yet built what is needed to take somebody on properly.

That is the central unsolved problem. An activity that does not survive its founder being unavailable is not a company, it is a job.

The sovereignty of the arrangement.

My structure depends on a registered office provider, payment institutions, infrastructure suppliers, platforms. Each of them can end the relationship. I have fallbacks, but the dependency is real.

Scaling.

The model works at one. I do not yet know whether it works at five. All the machinery described in this book rests on a rare combination: broad technical skill, commercial ability, and acceptance of risk. Finding that combination in somebody else, or organising it across several people, is still ahead of me.

Balance.

Two episodes of burnout, both self inflicted. I have learnt to close the computer, to take a full month off, to refuse a client in order to protect free time. But it is not natural, it is acquired, and it has to be relearnt regularly.

I have no method against burnout. I have only learnt to recognise it sooner.

The six decisions that determined everything

If I had to reduce this book to six decisions, these would be them.

One. Choose the least popular specialism, because the business market was looking for precisely that.

Two. Go into the engine room rather than into a comfortable post, to see how systems really work.

Three. Take an interest in the technologies everyone looks down on, because there is no competition on empty ground.

Four. Create the entity that invoices, to capture the value I was producing and somebody else was collecting.

Five. Give before asking, to enter a closed market through the only door still open.

Six. Refuse often, because the quality of a business is measured by the bad clients you did not take.

None of those decisions required significant capital, a network, a particular passport, or exceptional talent. All of them required accepting being alone for a while.

THE RULE THAT SUMS THEM UP

Look at where the crowd is going. Count.

If there are more people than places, that is not your ground, however attractive it looks.

Then look for the place where there are more places than people. It is always less prestigious, less talked about, less socially valued.

That is where you should go.

The closing word of this part

I have not built an empire. I have built a position.

A position from which I decide my clients, my pace, my subjects, where I live. Where my work is sold abroad while my costs stay local. Where I can close the computer for a month without putting anything at risk.

That is not the spectacular success of entrepreneurial stories. It is something else, and I believe it is better: it can be repeated.

Everything described in this book was done with a computer, a connection, a free education and a few hundred euros of formalities.

This is not a story about talent. It is a story about position.

◆

EPILOGUE

I will live here

> Why staying is a calculation and not a pose, and what I believe is still left to build.

I will finish with the least technical and most decisive part, because it explains everything else.

I have never considered leaving. Not one application, not one file, not one second of serious thought given to the question.

For twenty six years my holidays have been in the same place, in a region in the east of the country. That has never changed. I take the plane like everyone else, but to go there. Abroad, for me, means first of all the dozens of regions of this country I have not yet seen.

That is not closing yourself off. It is an order of priority.

The calculation, one last time

I will set it out one last time, coldly, because it is the only justification worth anything.

Leaving means starting an accumulation again. A degree to have recognised, a network back to zero, an administrative status dependent on permanent renewal, a multiplied cost of living, and often a debt taken on to get there.

I have watched closely what that configuration produces. Excellent engineers stacking up degrees at the same level because they cannot find a post. Experienced professionals accepting apprenticeship positions several levels below their real standing, be-

cause those positions are subsidised and therefore easier to get. Brilliant students who no longer dare take the slightest risk because their parents funded their studies at considerable sacrifice.

What all those trajectories have in common is the search for validation. You look for validation from a university, then from an employer, then from an administration. At every step you give up a little position.

Staying, with a structure that sells outwards, reverses all of it. I keep the local cost of living, the absence of debt and the family anchor.

This is not about going to find a level we already have. It is about going to find a purchasing power we do not have.

What I saw abroad

I have to repeat what I observed, because it contradicts the dominant story.

I worked for two years for foreign clients, with full access to their production environments. I then explored another market for several months, dealing directly with owners, directors, business advisers and heads of organisations.

I was disappointed.

Not by the people taken individually. By the gap between the image and the reality. Well oiled systems served by people who are often exhausted. Failing customer service. Slow administrations. Cascading subcontracting where every intermediary takes a margin

and adds nothing. A conformism that prevents anyone taking the slightest risk.

And a disconcerting readiness to sell sovereignty for easy money.

We are better here. Not in resources, not in organisation, not in presentation. In raw capability, in resistance to adversity, in depth of training, in resourcefulness.

> **WHAT IS MISSING, ON THE OTHER HAND**
>
> We have to be honest about what we do not have.
>
> We do not have a selling culture. We are taught to build, never to sell.
>
> We do not have a culture of the frame. Insurance, compliance, contracts, guarantees: everything that reassures an international buyer is foreign to us.
>
> We are not used to positioning ourselves high. The reflex is to place yourself as the one asking.
>
> Those three gaps are exactly what this book has tried to fill. They are not a matter of talent. They are a matter of information and method.

What is left to build

I believe this country is a largely blank page in many areas. There is everything to do.

We must export services, not people. Every piece of work sold abroad from here is foreign currency coming in, activity kept loc-

ally, contributions, taxes, and a skill that stays. Every person who leaves is the opposite.

We must build our own systems. We hand the core of our critical sectors to foreign vendors, with the consequences described in chapter 7. That was not an absurd choice at the time: systems were needed, and they existed elsewhere. But we now have to think about tomorrow, and current tooling makes that prospect infinitely more reachable than it was ten years ago.

We must stop looking for validation from outside. Not from rankings, not from foreign institutions, not from programmes handing out labels. Those things are useful as tools. They are destructive as goals.

We must pass things on differently. Not through large collective initiatives that run out of steam with the person carrying them, but through direct, concrete transfers. An access, a recommendation, half a day of sharing, a tool.

What I do not claim

I have no lesson to give and I do not claim to have found the formula.

There are things I have not solved, and they are listed in the previous chapter. My structure rests entirely on me. I have not managed to recruit. I do not yet know whether the model holds beyond one person.

There are subjects on which I am probably wrong and I will find out later.

And there is a great deal I have not done well: I have upset people, I have been too direct, I have confused being right with being useful.

This book describes a position and a method, not a model of virtue.

The last word

To those reading this and wondering whether they should leave, I will not tell them to stay. It is not my decision and every situation is different.

I will simply say: do the calculation before you make the choice.

Look at what you lose, in reality. The absence of debt, if you are lucky enough to have that. The cost of living. The family network. The social position already earned. The freedom to refuse.

Look at what you gain, in reality, not what you are told. Check with people who are there, asking them about the unpleasant things, not the photographs.

And above all look at the third option, the one nobody talks about: not choosing. Staying physically, selling elsewhere. It is technical, it is doable, it costs a few hundred euros and a few weeks, and everything needed to get there is in the preceding pages.

This is not about going to the market. It is about bringing the market here, and setting our own rules.

For my part, the question was settled a long time ago.

I will live here, and I will stay here. In between, we will do our best.

APPENDICES

Six tools you can use straight away

> The grids, sequences and checklists used in the book, gathered here to be applied directly.

Appendix A. Setting up a company abroad

Checklist, in order of execution.

Before you start

- ☐ I have already been paid at least once for what I intend to sell.
- ☐ I can name three specific organisations that have this problem today.
- ☐ I can fund twelve months of fixed costs with no income from this structure.
- ☐ I have a proper independent status in my country of residence.

Incorporation

- ☐ Choice of jurisdiction, based on the target market and not on tax.
- ☐ Choice of the simplest legal form available.
- ☐ Online registration, nominal capital.
- ☐ Professional registered office address, with mail forwarding.

- ☐ A backup registered office solution identified.

Identity

- ☐ Dedicated domain name, registered for several years.
- ☐ Professional email on that domain.
- ☐ A site setting out the offer, the legal notices, the contact details.
- ☐ Professional profiles aligned with the positioning.

Finance

- ☐ Main business account in the jurisdiction.
- ☐ Account with an international payment institution.
- ☐ Third backup account.
- ☐ Capped virtual cards for subscriptions and trials.
- ☐ Invoicing and payment tracking solution.

Armour

- ☐ Professional indemnity insurance taken out.
- ☐ Registration with the data protection authority.
- ☐ Standard confidentiality agreement, ready to sign.
- ☐ Data processing agreement, ready to attach.
- ☐ Standard service contract, with a reversibility clause.
- ☐ Security architecture note, one page.

Leverage

- ☐ Registration for the startup programmes of the infrastructure providers.
- ☐ Registration on the deal aggregation platforms for founders.
- ☐ A calendar of credit expiry dates and milestones to reach.

• • •

Appendix B. Qualifying a prospect

Scoring grid. Add up, engagement threshold from sixty points.

Criterion	Observation	Points
Sector dependent on a critical information system	Yes	10
Big enough for a budget, too small for the large integrators	Yes	10
Old technology detectable from outside	Yes	15
Job advert open for a long time on a rare skill	Yes	20
Key technical profile signalling availability	Yes	20
Regulatory deadline or audit coming up	Yes	15
Recent public incident	Yes	10
Decision maker identified and reachable	Yes	15
Recent growth or fundraising	Yes	10

Knockout criteria

Whatever the score, you do not go if:

- no decision maker can be identified;
- the organisation belongs to a group whose systems are centralised elsewhere;
- a supplier has just been engaged on the same scope;

- no pain signal is detectable, only a theoretical technical mismatch.

> REMINDER
>
> The best buying signal is human, not technical.
>
> An organisation puts up with an old system for twenty years. It moves the day the person who maintains it leaves.

• • •

Appendix C. Prospecting sequence

Three messages, spaced out. Structure rather than exact wording: the wording must come from you.

Message 1, day zero

Subject. An observed fact, never an offer. Six words maximum.

Body, four short paragraphs:

1. The observed fact, precise and verifiable, with how you saw it.
2. The possible business consequence, phrased without drama and without judgement.
3. What you are offering, bounded and concrete. A deliverable, not a meeting.
4. A closed question.

Signature. Full identity, entity, verifiable contact details.

Message 2, a few days later

Three to five lines. You add something new: a second observed signal, a sector detail, an order of magnitude.

You never repeat the first message.

Message 3, one to two weeks later

Two to three lines. You close cleanly: I remain available, I will stop contacting you.

That message is the one that produces the most replies.

Absolute prohibitions

- Any wording that could apply to a thousand recipients.
- Any criticism without a solution in the same sentence.
- Any information obtained other than from a publicly accessible source.
- Any breach of sending quotas.
- Any recontact of a person who has opted out.

• • •

Appendix D. Auditing a legacy system

Interview and observation grid, to be run in three hours.

Continuity

- When was a backup last actually restored?
- How long to bring the service back after a total loss?

- Who knows how to do it? How many people?
- What happens if that person is unavailable for three weeks?

Dependency

- How many people can modify the business core?
- How long have they been in post?
- Do you hold the source code, for every component?
- Are there components for which you have neither the code nor the contract?

Exposure

- Which services are reachable from outside?
- Which versions are out of support?
- Are there remote access routes left open for convenience?
- Have the default credentials been changed?

Control

- Is there a representative test environment?
- Are there automated tests on the critical paths?
- Is deployment automated and documented?
- Is there monitoring with alerts?

Structure

- Is the customer specific logic hard coded?
- How many different versions are in production?

- How long does it take to onboard a new customer?
- What share of the code is never called any more?

Reporting back

For each finding: the fact, the business consequence, the severity, the effort to fix.

Then three lists: to do this week, to do this quarter, to decide.

And one extra list, the most appreciated of all: what must definitely not be done.

• • •

Appendix E. The almost zero cost kit

What a one to three person structure can assemble, legally, by making use of the existing schemes.

Need	Solution	Cost
Servers and databases	Startup programmes of the infrastructure providers	Credits granted in tiers
Development assistance	Developer platform programmes	Included or credited
Continuous integration	Free quotas of the platforms	Free within quota
Transactional sending	Dedicated service or professional mailbox interface	A few euros a month
Site hosting	Continuous deployment platforms	Free within quota
Domain name	Low cost registrar	A few euros a year
Professional email	Domain forwarding or office suite	Free to low cost
Office suite	Shareable family plans	Low annual cost
Scientific documentation	National research portals	Free
Advanced professional research	Trial periods, referral offers	Temporarily free
Professional insurance	Specialist insurer	A few dozen euros a month
Data compliance	Registration with the regulator	A few dozen euros a year

Compulsory discipline

- Everything is containerised, so movable within hours.
- No proprietary component on the critical path.
- Real cost excluding credits estimated and kept up to date.
- Expiry calendar tracked.
- Spending caps and alerts active on every service.

• • •

Appendix F. Glossary

Open source analysis. Collecting publicly accessible information, with no unauthorised access and no intrusive testing.

Neuro symbolic approach. Combining deterministic processing, which produces verifiable structural facts, with a language model, which produces text and proposals. Each on what it is good at.

Abstract syntax tree. A tree representation of the structure of a program, produced by parsing. Cannot hallucinate.

Implicit cycle. A mechanism in certain old languages where the system automatically triggers processing as records are read, with no explicit loop in the code.

Facade. An interface layer placed in front of an existing system, through which calls pass, allowing gradual redirection to new components.

Strangler pattern. A gradual modernisation strategy: you surround the existing system, move function by function, and remove the old one once it is no longer called.

Pseudonymisation. Replacing an identifying element with a token, keeping a mapping table separately. Reversible.

Anonymisation. Permanent removal of the link with the person. Irreversible.

Selective retrieval. A technique consisting of sending a model only the relevant fragments of a corpus, rather than the whole corpus.

Reversibility. The client's ability to take back his system without his supplier. Covers the code, the access, the documentation and the procedures.

Compliance airlock. A device that masks locally before sending to an external service, and unmasks locally on return.

Characterisation test. A test that records the current behaviour of a system and checks that it does not change, without judging whether it is correct.

Batch processing. Deferred and grouped execution of operations, as opposed to immediate processing. Still the real foundation of most large business systems.

• • •

These tools are only worth anything through use.
Take them, change them, and above all apply them.

◆