From Zero to AI Agents That Do The Work

From Zero to AI Agents That Do The Work

How to build them and put them to work

Floris Knol

Floris Knol

ISBN 978-94-0394-429-6

Cover design: Floris Knol

First edition, 2026

Published via Bookmundo

AI tools were used in making this book. The choices, the structure and the wording are the author's; the examples and the figures come from work he did himself.

Contents

PART III — WHEN IT GOES WRONG

PART IV — IT BECOMES AN ORGANIZATION

PART V — OUT IN THE REAL WORLD

IN CLOSING

Foreword

Something has changed in how work gets done, and almost nobody can tell you precisely what.

You hear that AI changes everything. You see examples that are impressive and that you can then do nothing with. You read posts by people who know, courses that promise, and articles that after three paragraphs tell you the main thing is to start. And then it is Monday morning, you are looking at your own work, and nothing has changed.

This book was written by someone who did do it, and it is about what he learned along the way.

What happened here

What this book is about is an environment where agents do work by themselves. Not one assistant helping you, but several running at once, each with an assignment, a limit, and somewhere to record what they did. And that in turn is not an end point: it is the step toward an organization that runs on AI rather than one that uses it.

I built that, from nothing, with what is simply available now. What made the difference was the way of choosing, and I found that on the way.

The unusual part is not the speed. It is that it got easier every day instead of harder.

With almost everything it is the other way round. What you build gets heavier: more parts, more exceptions, more things that can break, and so less speed. Here the opposite happened, and that was not luck and not talent. It came from a small number of moves that looked like nothing on the day.

Those moves are what this book gives you.

A report from along the way, with the patterns pulled out.

I wrote everything down as it happened: what I did, why, what it cost, and what went wrong. Out of that pile come about ten patterns that kept repeating. Those patterns are the book; what I happened to be building is the example.

That distinction matters, because otherwise half the readers are watching instead of doing. What I built was technical, but none of the patterns is about technology. They are about the order in which you do things, about where your time goes, about what you can hand to a machine and what you cannot, and about how you notice you are working in the wrong direction. A bookkeeper, a photographer, a lawyer and a builder all run into them. Which is why every pattern also comes with an example that has nothing to do with software.

And you can do it yourself. Every number in this book comes from work that was really done, and next to every number is how it was measured. You run the exercises on your own screen, and then you see whether you come close. That is rare in this subject and it is deliberate: what you have seen work yourself, you do not have to take from me.

Including what went wrong. There is a day in this book on which I drew the wrong conclusion three times in a row, a moment when my own tools lied to me, and a measurement that proved my way of measuring was broken rather than the thing I was measuring. Those are not in here out of modesty. They are in here because that is where most of the learning was.

Where this goes

What starts with one person and an empty machine ends in something that takes on work by itself in the real world.

Put like that it sounds like boasting, and I understand that. But it is literally the shape of this book, and not one of the steps in between is spectacular. It helps to see them in advance, because then you understand why the beginning is so small.

It starts with making one thing cheap. One task faster, one check that does itself, one mistake you do not make twice. That is the first half of the book and it is plain craft.

Then a hundred run at once. Not because that is impressive, but because the question changes as soon as there is more than one: what you take in at a glance with one has to be written down with a hundred, and what you do yourself with one has to happen without you with a hundred.

Then something goes wrong, and it corrects itself. Something that notices it is stuck and stops. Something that sees the same mistake coming back and changes the instruction rather than the mistake. Something that has a limit and does not cross it, not even when it would be convenient.

And then it becomes an organization. That word is not meant grandly. An organization is something with a goal nobody has to hand it again every morning, a budget that can run out, roles that know about each other, and a memory. Once those four are there, it is no longer a tool waiting for you.

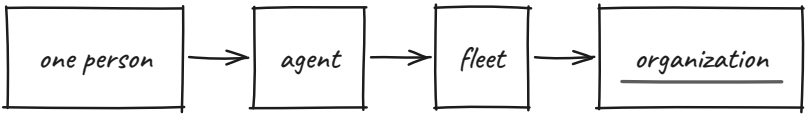
And then comes the part almost nobody mentions. Such an organization does things in the real world. It buys. It signs up to services and takes on obligations by doing so. It hires people — because the idea that people stop being involved is a fantasy: there is work a human has to do and work a human has to sign for. It adjusts its goals, and sometimes it scales itself up.

Which means it has to **take part in the formal world.** An accountant turns up. An audit turns up. There is a standard a process has to meet, and somebody asking: show me who approved this, when, and on what basis.

That is not an awkward extra but the condition. An organization that cannot show what it did is not autonomous — it is unsupervised. The difference between those two words is exactly the difference between something you may use inside a company and something you can only try out at home.

And the good part is that they are the same things. The recording you need in order to learn from it is the recording an auditor wants to see. The limit that stops the money running out is the limit a controller asks about. Do the one well and you get the other for free — skip it, and you will have to build it later at the worst possible moment.

I am not yet at the point of an autonomous organization. I am somewhere in the middle of it, and the book says exactly where. But the direction is clear, and every step toward it is one you can take today.



each step makes the next one possible

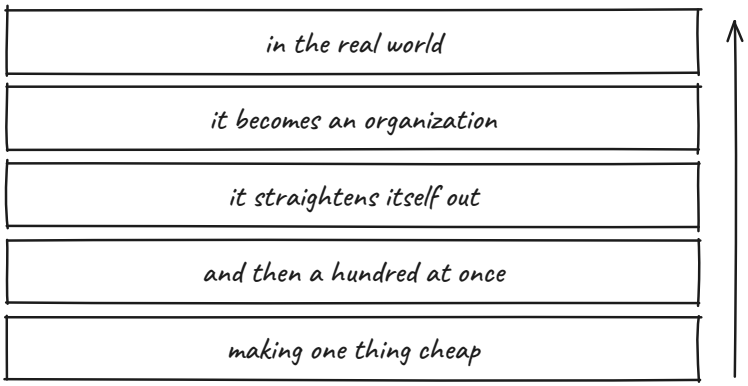
The shape of this book: what begins with one person ends with something that takes on work by itself.

How to read it

The five steps above are the five parts of the book, in exactly that order: making one thing cheap, then a hundred at once, then what happens when it goes wrong, then it becomes an organization, and then the real world. After that comes one more chapter that looks beyond where I stand — the only one that is not yet practice.

Every chapter comes down at least once to something you can try tonight. It sits in the text itself, with why it works and where it goes wrong. You can read past them; you can also do all of them, and then you will have built something by the end while reading.

And every chapter closes with one line: **what this made cheaper.** Those lines stack. Read them back one after another and you see what the book is actually about.



The five parts rest on each other: each one makes the next possible.

Why now

Because you are not late yet.

That is not a sales line but an observation. The things that make this possible are all here now, they are cheap, and almost nobody uses them together. That is exactly the moment when somebody who does builds a lead that money cannot close — because the lead is in what you have stacked, and stacking takes time nobody can skip.

In two years this will be ordinary. Not yet.

Introduction

I am a programmer. That is the easiest way to dismiss me, so let me put something next to it straight away.

I also spent years supporting people into work. People without paid work, people with autism, people who had been outside employment for a long time, people for whom the ordinary route did not work. That is different work from building software — different words, different outcomes, and nobody confuses the two.

In that work I unlearned and learned four things, and none of them was about computers. **Preconceptions do not help**, because whoever walks in with a file in their head sees confirmation of what they already thought. **Building up does**, step by step, because a step that fails only tells you something if it was one step. **Repeating and stacking**, because what works once can be luck and what works three times is a floor to stand on. And **making it systematic and predictable** — the same order, the same words — because that removes all the noise that has nothing to do with the matter at hand.

There was one more, and it turned out to be the most important: **give facts, not assumptions and not opinions.** *"He is not motivated"* is an endpoint. *"He did not come on the last three Tuesdays"* is a starting point. The first closes; the second opens.

When I started working with AI, I did not have to learn anything new. I did what I was already doing, and it worked immediately.

That surprised me more than it should have. A model has no history with you, cannot guess what you mean, and does not read between the lines. Everything you do not supply as a fact, it fills in itself. Exactly the same limitation as in a room with someone you are meeting for the first time — and therefore exactly the same method.

That is how this book came about. Not from an idea about AI, but from noticing that three worlds with nothing to do with each other ask the same questions. I did not invent the patterns in it; I recognized them. That does not make them new. It makes them usable in places where you would not go looking for them.

What *is* new about it is not the idea. **It is that something now exists that carries them out while you are not there.**

This book is about what that changes, and what you have to do to make it hold.

And that is why there is not a single trick in it. What works for a few months and is wrong after that teaches you nothing — you still do not know *why* it worked while it worked.

What is in it are concepts and patterns. They are about the order in which you do things, about where your attention goes, about what you can hand over and how you can tell. Such things are not tied to a program, a supplier or a version. **They held before this tooling existed, they hold now, and they will still hold once everything you use today has been replaced by something better.**

That is the promise of this book, and it is the only one I dare make: you will still have use for it in a few years, in whatever environment you work.

And yet every chapter comes down once to something you can do tonight. That looks like a contradiction with the paragraph above and it is not — but it deserves its own explanation, and that comes next. Anyone who would rather start straight away can skip it and go to chapter 1.

How the instructions in this book work

This chapter sits at the front because it is about the form and not about the content. You can skip it and start on chapter 1; you will then simply come across a few things and wonder why they look the way they do.

In short: every chapter that lends itself to it comes down once to something you can do tonight. What follows is why that is there, how to use it, where the examples come from, and what was changed about them before they went into this book.

What a worked example does

A worked example is the shortest route from a concept to something happening on your own screen, and that is exactly why the concept sticks. You have seen it work once; the second time you come up with it yourself.

So every chapter that lends itself to it comes down once to something you can do tonight: what you do, what you literally type, what comes back, and where it goes wrong. Allow five to twenty minutes. You can skip them and the book still holds; you can do all of them, and then you will have built something by the end while reading.

The examples are written for one program, and that is Claude Code. Not because it is the best and not because it has to be, but because something has to be there that you can type out. An example that says

"ask your AI for something like this" is not an example but an evasion: it hands the only hard part — writing down precisely what you want — back to the reader.

With any other provider it works the same way, in the same words. What these examples ask of you is never a button, a menu or a setting; it is an instruction in plain language, and plain language is precisely what all those programs have in common. *Where* you type it differs, and changes every six months besides. *What* you type does not.

And this is not an introduction to using such a program. That you can type an instruction and get an answer is assumed here; there are manuals for that and they are better than anything I would write about it. What begins here is the step after that, and it is a different step from what most people expect.

That step is: from something that answers to something that works while you are not there. Not a conversation in which you make the next move every turn, but an instruction that gets carried out, checked and finished without you in between. The difference between those two is not one of degree. The moment you stop watching every turn, everything you were silently doing has to come from somewhere else: what exactly should happen, where it stops, how you can tell it is right, and what happens when it is not.

That is what this book is about, and where the examples take you. The first one is small and you still have your hand on the button. Each next one takes a piece of you out of the loop — until there is something that accepts an instruction, carries it out, checks itself, and comes back only when it is done or when it cannot work something out.

All the examples are about the same construction company. One world that grows with you: jobs, clients, staff with certificates, equipment with inspections, invoices, a diary, tenders — and twelve rules it is supposed to follow. Small enough to look inside, real enough to go wrong somewhere. The first example has you describe that folder; after that every example works inside it. By the end you will have built one thing rather than twenty loose experiments.

And that folder is deliberately not in order. Every rule is broken exactly once: a number that is missing, a job referring to a client that does not exist, somebody who did work with a certificate that had expired a month earlier. That is not sloppiness but the entire point. Building a check against something with nothing wrong with it yields green without telling you whether it works — and that is precisely the mistake this book devotes a whole chapter to.

And everything you see coming back in those examples really came back. Every example in this book was run in that folder, with exactly the instruction printed beside it. What appears under *what came back* is what came back — not a tidied-up version, not something I retyped because it looked better.

Two things I did do to it, and it is fairer to say so. I **shortened** the pieces of output so they fit a book page: never more than eight lines and never wider than the column. And I removed the paths and dates from my own machine, because on yours they are different anyway. **Nothing was reworded.** Where an answer was clumsy it stands clumsy; where it was wrong the mistake is still there — and in a few cases that mistake became the point of the chapter.

One thing I did not really do, and you should know it. In the last chapters the examples reach outward: an email to a client, a notice to the city, an entry in a set of books. Those instructions were really carried out, but on a practice rig — no email was sent and no system belonging to anyone else was talked to. What comes back is what such a recipient would answer, including the times they do not.

At your end it does go out for real, and that is exactly why those chapters are about limits and signatures.

That is deliberate. An example polished until it looked good would wipe away precisely the experience you are about to have yourself.

Do not copy them. Do them.

There is one thing I want to say with emphasis, because it is the difference between a book you have read and a book you got something out of.

The instructions in this book are not recipes. They are there so you can see what such an instruction looks like and what it produces — not so you can type them out and read on. Type them out only and you get something much like what is printed, and you learn almost nothing from it, because it already held.

And note: **you will not get literally what is in this book anyway.** Ask the same thing twice and you get something comparable twice, in different words and sometimes in a different order. That is not a fault but how this works — and it is the second reason not to compare but to try. What has to hold is the outcome, not the wording.

Change them. Put your own rule in, your own folder, your own number. Ask for something other than what is written. Leave something out and see what happens. **That is where the learning is**, and it is also exactly what happens the moment you apply this to your own work: there, nothing is ever the same as here.

That is why almost every example ends with a second round. First the instruction, then what came back, then what was not right about it, then an adjusted instruction with a new answer. That second round is not there for completeness. It shows that the first attempt is rarely right, and that adjusting is not a sign that you do not understand it but the way this works.

Do one. Make a mess of it. Adjust it. Then you have it.

And they change in kind along the way

What you do changes through the book, and that is the whole movement this book describes.

At the start you type an instruction and read the answer. You take the next step every time. That is where almost everyone stops, and it is fine too — there is a lot to be gained with just that.

A little further on you record what you just did, so you never have to explain it again. You are no longer carrying out but setting up, and you notice there are things you never type again.

After that you put things in place that call themselves. A check that runs when something has been added. A limit that holds back first and asks afterwards. You are not there at that moment, and that is the point: the next step is no longer taken by you but by the situation.

And by the end you are no longer putting instructions in place. Something is running that you once set up. You check in, you look at what happened while you were not there, and now and then you adjust something. Nothing more. **It keeps going.**

That is why an example in the last part looks different from one in the first. There is no *type this* there any more, but *look at what happened*. Not because there is nothing left to do — but because the doing has moved somewhere else.

I say that here once and not again at every example. A caveat that comes back a hundred times gets skipped a hundred times.

The example folder

Every instruction in this book plays out in the same folder. Before you start on chapter 1 that folder has to exist — and that takes one instruction.

Why a building firm

What matters is work with agreements in it, and every organization has that — a school, a catering company, a bookkeeping firm. A building firm is the concrete example this book uses.

Inside it sits the web that breaks administration. Jobs point at clients, clients at invoices, invoices at hours, hours at people, people at certificates with an expiry date. That is exactly what an agent can walk through and you cannot.

And you do not need to know the trade. Everyone understands that a certificate has to be valid before you climb the scaffolding. Replace building firm with whatever you do — the rest of the book reads the same.

The instruction

Make an empty folder, open your agent in it, and type this:

> *Create the administration of a building firm with five people on the payroll in this folder. Use folders for jobs, clients, staff with certificates, equipment with inspections, invoices, timesheets, quotes, purchasing, subcontractors, a diary and tenders — around thirty in total, with a few documents each. Write everything as markdown with fields at the top. Add a* `rules.md` *with twelve rules the administration is supposed to follow: numbering of jobs, valid certificates for the work someone does, inspections of equipment, a limit for extra work, and so on. Have each rule broken exactly once, spread across the folder, without saying anywhere where.*

Reckon on a few minutes. You get no progress bar; you see folders appear and then a summary of what has been made.

What to look at when it is done

Four things, and all four can be counted:

- **around thirty folders** at the top level, including the eleven from the instruction
- **something like a hundred documents**, spread across those folders
- `rules.md` **with twelve rules**, numbered
- **fields at the top of every document** — a job has a number, a client, an amount and a date, not running text you have to fish them out of

If that is right, you can go on. You are not going to look for the twelve breaches: that is the work of chapter 1 onwards, and the agent deliberately did not write them down.

What you get back is not what I have

Your folder looks different from mine. Different names, different amounts, different order, and probably a few folders I do not have — or the other way round.

That is good, and it is the reason this works. The patterns in this book do not hang on a filename. They hang on the shape: there are agreements written down, there is something that does not keep to them, and there is a way to find that without walking through everything by hand.

What does differ are the **numbers in the outcomes.** Where this book says a check found nine, yours may find seven or eleven. So do not compare the number but the shape: did you get a list with locations, or did you get "looks fine"? That difference is what chapter 4 is about.

When it goes wrong

There are three ways this does not work first time, and none of them calls for starting over.

Too few folders. Some agents make ten and stop. Ask for more: *"add another twenty, same setup"*. Adding is cheaper than redoing, and it is straight away the first time you notice how a conversation like that works.

No breaches. An agent that makes a tidy administration has ignored the last part of the instruction — that happens, because it is an odd thing to ask. Say it explicitly then: *"now add exactly one breach per rule from rules.md, spread out, and write down nowhere where"*.

Running text instead of fields. If you get documents that read like a letter, ask for the shape: *"put the fields at the top of each document on their own line, like* `amount: 1850`*"*. Without that shape a check cannot rely on them later.

And if you would rather do it over anyway: empty the folder and type the instruction again. Nothing is lost — this is exactly the kind of work that ought to be cheap, and that is what chapter 10 is about.

This is the only step in the book you do just once. Everything after it builds on this: every example works in this folder, and at the end there is something that checks itself.

PART I

Making one thing cheap

1. The tools first, then the thing

It is two minutes past eight in the evening. There is nothing.

Four minutes later there are four things on disk. Not one of them is the thing that had to be built.

There is **a tool** to make it with. There is **a way to check** whether what gets made is right. There is **a list of what still has to happen**, in order. And there are **examples**: a few small things that already work, to copy from.

The thing itself is not there. That comes tomorrow.

What happened there

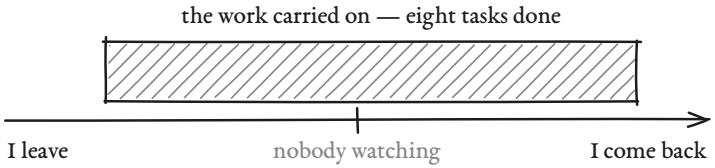
Someone who wants to make something makes it. That is not strange: you have an idea, you get to work, and the tools turn up when you need them. Almost everybody works that way, and so had I, always.

What happened that evening was something else. The first move was not *start* but **make starting cheap.**

That took an evening and produced nothing that evening. At the end of the day there was nothing to show. There was only a place where making something would take less time than it would anywhere else.

That is the first pattern in this book, and it is the one all the others rest on:

> **Before you make the thing, make the things that make the thing cheap to make — and cheap to check.**



What happened between walking away and coming back.

Why that evening existed at all

There is a reason behind it, or this is just an evening with a good habit rather than the start of something.

The cause was size. Not of the project — of the tooling. A modern development stack has grown so large that no short loop fits inside it any more. So much sits between changing something and knowing whether it is right: building, loading, waiting, starting, dependencies that want something else. For a person that is tedious. For a machine doing it a hundred times in a row it is fatal.

And that is the whole problem in one sentence: with a stack like that, no short feedback loop for a machine can exist. Everything in this book about trying, measuring, trying again and letting it check its own work rests on a loop short enough to run a hundred times. If it is not, all of it is theory.

So that evening was about shortening that loop, and what came out of it was whatever was needed to get it short.

Did I expect it to work? Yes, and that is not bravado. I had watched it happen on a small scale in the preceding months — enough to know it ought to be possible. Not *that it would succeed* but *that it must be possible*, which is exactly the difference between a gamble and an expectation resting on something. This is the same move as the tallying on that scrap of paper in chapter 3: measure first, then be sure.

This is not about software

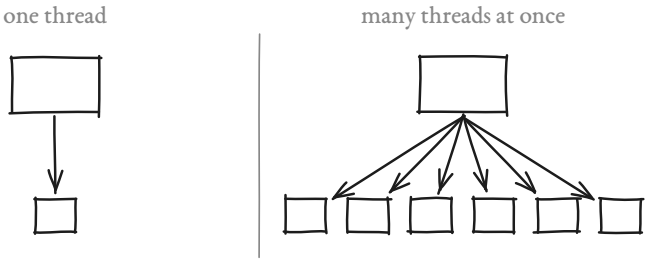
It may sound like something for people who write programs. It is the other way round: it simply shows up there faster, because everything can be measured.

A photographer who takes two thousand frames on a job can look through all of them one by one. Or he can spend half a day first on fixing his lighting and his processing, after which every job starts at eighty per cent. That half day produces nothing that day.

A lawyer can write every contract from scratch. Or he can build a set of clauses he reuses, and then writing a contract has become a different activity from what it was.

A teacher can give the same explanation every year and every year notice again where students stumble. Or he writes down once *where* they stumble, and changes the explanation there.

In all three cases the move that counts is the move that produces nothing that day. That is not a coincidence; it is the definition. A move that pays off immediately usually pays off only that once.



One person, one thread — and one person, many threads at once.

Try it yourself · Ask what is actually in there

What you do. Open the example folder and ask what it holds. Build nothing, change nothing — just look. This is the shortest version of what this chapter describes: you type one sentence and get back something you would otherwise have spent a morning on. Allow two minutes.

TYPE THIS

> This is the administration of a construction company. Describe in ten lines which business processes you can see here, then name three things that stand out to you.

What you see. A list of processes you had never laid out like that yourself — from acquisition and estimating to quality assurance and aftercare — plus three remarks about what stands out. **Watch the third remark.** The first two are usually what you already knew; the third is regularly something you had never looked at.

WHAT CAME BACK

```
integrations 17  jobs 15      execution 14
timesheets 12    diary 11     development 11
safety 11        invoices 9   estimates 8
stores 8         staff 8
work-preparation 8
... 49 folders, 254 files
notable: 1 gap in the job numbering
notable: 1 job with no materials
```

Where it goes wrong. Ask for a number of lines. Without "in ten lines" you get two pages, and then you are reading everything after all. **Ask for something that requires a judgment.** "Describe what is in here" yields a table of contents; "name three things that stand out" forces a choice, and the value is in that choice. **And do not simply believe the list.** What gets listed is what is in the filenames, not what actually happens — and that difference is exactly what chapter 14 is about.

THEN LIKE THIS

> For each of those three things that stand out, name the file you took it from, so I can check it myself.

AND THEN THIS

```
gap in the numbering
  -> jobs/, 15 files, ENJB-2026-005 missing
job without materials
  -> jobs/ENJB-2026-003.md
unknown client
  -> jobs/ENJB-2026-006.md
```

The unusual part

What stood there later I could never have built alone if I had simply started. But that is not the interesting number.