Glad you downloaded my book. I hope it provides you with valuable practical insights."

If you found it helpful, a small contribution would be greatly appreciated. You can donate via the QR code below.

Scan om te betalen

Or buy a hard copy via www.consultancyenzo.nl

THAT IS NOT WHAT I MEANT!

Stop taking notes and start solving problems

Michiel Lensink

© Michiel Lensink, 2026

All rights reserved. Use the knowledge in this book at your own discretion, but copying the literal text is not permitted without written permission.

First edition: September 2026

Want to buy an (extra) copy of this book ? www.consultancyenzo.nl

ISBN (hard copy): 9789465545851

INDEX

Why I Wrote This Book **7**

About Information Analysis **10**

What Is Information Analysis? 11
Why Projects Fail 13
Information Analysis Is A Profession 16
Is Information Analysis always Necessary? 18

Information Analysis in 3 phases **20**

Phase 1: From Need to Direction 23
An Exporatory Conversation 25
Know Who You Are Working For 26
The Need Behind the Request 30
Designing (Visually) in Broad Outlines 33
Breaking It Down and Prioritizing 37

Phase 2: Detailing the Specifics 41
Detailing the Building Blocks 41
The Joint Review 46

Phase 3: Execution and Adjustment 51
From Analysis to Reality 51
Gathering Feedback 53
Aftercare is Also Part of the Process 56

A Detailed Example 58

Conversation techniques **64**

Drawing 65
Behaviour-Oriented Probing 68
The 5 Whys 70
Focus on Emotions 73
Sketching a Future Vision 76

Ask About Previous Solutions 79
Suppose It Failed 82
Summarizing and Verifying 86

Techniques to Structure Your Work 90

The Flowchart 92
MoSCoW: When Everything Seems Important 99
Personas: Because "The User" Doesn't Exist 102
INVEST: The Quality Check for Building Blocks 105
Artificial Intelligence: Your Assistant 110

Basic Technical Knowledge 113

API's: How Systems Talk to Each Other 115
Databases: The Memory of the System 117
Environments (DTAP) 118
Logging: The Black Box 118
Frontend, Backend and Microservices 119
And Then You Also Have… 120

Some Last Tips 121

The Forgotten Requirements (It works, but…) 122
The Art of Saying 'No' 123
Where do You Keep the Truth? 123

Finally : It's Your Turn Now 125

Sources & Reading Tips 127

Why I Wrote This Book

Ever worked on projects? You worked hard, and everything went (mostly) according to plan. Yet the outcome isn't what was intended somehow. The outcome falls short of expectations, the collaboration didn't go smoothly, or so many adjustments were made along the way that the original goal has become barely visible.

That is exactly where analysis makes the difference. If you blindly trust what someone says they want, or assume too quickly that you understand the question, you often end up with something that is well-intentioned, but solves little. This book is for everyone who wants to prevent that. Whether you are a subject matter expert, hold conversations with customers, or just often find yourself thinking: "Couldn't we have done this smarter?"

I started my career in software development in the early 2000s. I was mainly busy programming. I received assignments and plans on my desk, but I had little insight into the why. If the result didn't work, or the customer wanted something else after all, we adjusted it. And again...

When I made the switch to an analyst role, I thought: now I'm going to do it better. But in the beginning, I made exactly the same mistake as my predecessors. I neatly wrote down everything the customer said, nodded during meetings, but sat behind my desk afterward still having questions.

It finally clicked when a customer said to me: "You are not an analyst; you simply take notes." That wasn't nice to hear, but he was right. I noted what was said, but I didn't ask critical questions. I didn't sufficiently understand why something was important and simply translated wishes into documents, instead of solutions.

From that moment on, I changed course. I dove into the theory but noticed that many books were unnecessarily complicated or technical. That is why I developed my own approach: simplicity, working visually, and digging deeper until you reach the core. That turned out to be much more effective.

What You Can Expect From This Book

In this book, I share what I have learned. No thick, theoretical manuals, but practical tips and simple techniques that really work. Whether you are naturally analytical or not, this book helps you to think more focused. That starts with listening, digging deeper, and understanding what is going on. If that succeeds, collaborating suddenly becomes a lot more enjoyable, and results improve.

The focus lies on analysis in a flexible environment, where you can always adjust. However, the insights are applicable more broadly; everywhere people work together, this book applies. As my background is in IT, many examples come from the software world. But don't let that scare you: I have written it in such a way that you can easily apply the content to your own situation, whether you are building software, want to improve a process at work, or are organizing an event.

This Book Is Written Simply On Purpose

You will notice that I choose short sentences and have tried to avoid difficult words. That is not a coincidence, but a conscious choice. In a profession like information analysis, things are quickly made unnecessarily complicated.

Just compare:

"Within the context of agile development, it is crucial that requirements are evaluated iteratively to be able to respond adaptively to changing business needs."

Or simply:

"Stay critical of requests, because what the customer says today, can change tomorrow."

I choose the second.

About Information Analysis



About Information Analysis

There are many misunderstandings about information analysis. That it is only for IT professionals, that it takes days of work, and that it is unnecessary in modern ways of working.

In this chapter, I'd like to clear up these misunderstandings. Information analysis is applicable everywhere, whether you are an IT professional or an office manager who has to organize an event. Thanks to analysis, you understand why and for whom you are doing something, and the what and when then do always improve.

In this chapter, we look at exactly what the profession entails, why projects without analysis fail so often or take longer than necessary, and when you use it.

What Is Information Analysis?

You take something that is still vague, unclear, or half-expressed, and you make it usable. A wish, an idea, or a complaint lands on the table:

- "It all needs to be a bit easier."
- "We want to be able to handle this ourselves from now on."
- "There just needs to be more overview."

Everyone nods, but you think: "Why? What exactly do we mean by this? And what specifically needs to happen to achieve that?"

That is exactly where your work begins.

No Lists, But Meaning
A good analyst is not a note-taker who simply writes down what someone says. You listen, you dig deeper, and you unravel what is really meant.

- You translate vague statements into concrete goals.
- You turn words into behavior.
- You turn feelings into action.

Someone can tell their story perfectly, without it really working in practice. What sounds logical to one person, can be disastrous for the work process of another. That is where your craftsmanship comes in: translating nice words into a working system or process.

Example: "It Needs To Be More Customer-Friendly"

A sales manager says: "The system needs to be more customer-friendly, because customers are getting lost."

The beginner analyst notes: "Requirement: System must be customer-friendly." A developer can't work with such a requirement. As an experienced analyst, you dig deeper:

- "What is happening now that makes you say it isn't friendly?"
- "When does the customer get stuck?"
- "What do you want them to be able to do?"

After a few questions, it turns out that new customers can't find their invoice on the website and then call the company.

The real request is therefore not "customer-friendliness," but: "After logging in, a customer must be able to open their invoice within three clicks, without help from support."

But Isn't That Business Analysis?
The terms business analysis and information analysis are often used interchangeably. Yet, there is a difference in focus.

Simply put, business analysis looks at the 'why', and information analysis looks at the 'how'.

	Business Analysis	**Information Analysis**
Focus	Goals, processes, strategy, money	Requirements, sources, data, solutions
Result	Opportunities, business cases	Clear and testable solutions
Core Question	"Why are we doing this and what does it deliver?"	"What must the solution do?"

My definition: You do both. In this book (and in practice), that line blurs. As a good information analyst, you also need to understand *why* something is needed before you can figure out *how* it should work. You don't have to write business cases, but you do need to understand how something fits within something existing, like a way of working.

That is why in this book I use the term **information analyst** for the entire spectrum: from understanding the value to working out the behavior.

Why Projects Fail

Everything went according to plan. You had gathered the requests, the team was running smoothly, progress was tracked neatly, and the deadline was met. Every step delivered visible results.

And yet… no one is really happy with the end result. Not the user, not the organization, and actually, not even you.

Frustrated, you shout: "But wasn't this exactly what you asked for?"

The responses:

- "Yeah, we thought this would be useful, but now that we actually use it, it really isn't."
- "It does exactly what we agreed on, but it doesn't really fit our process."

Before you know it, you are in 'phase 2,' which is actually just an expensive correction round with more frustration and less budget for actually delivering value.

So What Goes Wrong Here?
Whether you work flexible or with a project plan set in stone: the problem often lies not in how you build, but in what you assume. Often, building is based on an assumption that hasn't been investigated enough, or a request that has been taken too literally.

This is exactly why information analysis is not a 'chore,' but the foundation of success. Analysis does not start with the solution, but with the questions: *Why* and *For whom?* Only when you have those answers, do you start thinking about the *What* and *When*.

Good analysis delivers three things:

1. **It prevents waste:** You build the right thing immediately. Expectations are aligned beforehand instead of corrected afterward (which is always more expensive).
2. **Concrete results:** It delivers requirements that you can build and test directly.
3. **Trust:** Perhaps the most important one. Trust with your customer that they are understood, and trust with your team that they are doing the right thing.

Example: The Travel Pass

A few years ago, I joined an ongoing project where the atmosphere was a bit tense. The initial request seemed crystal clear:

"We want employees to be able to request new travel passes quickly and easily themselves."

It sounded logical. The team built it, it worked, and it was delivered. But a week later, complaints poured in:

- "Colleagues who already have a pass are now requesting another one just to be sure they get one. That isn't allowed, so now we have to check everything manually for duplicates."
- "People now choose between 'unlimited travel' and a 'fixed route subscription' themselves, but they always pick the most expensive option, even if it doesn't fit their job role."

The solution worked fine (you could request a pass quickly), but the administrative chaos had doubled.

If someone had dug deeper at the start:

- "Who is this meant for?" → Only people without a pass.
- "Are they allowed to choose anything?" → No, that depends on their job role.

Then the need would have been formulated very differently: "We want to offer employees without a travel pass a subscription type that fits their job role." That would have immediately resulted in a better system and prevented errors in administration.

Information Analysis Is A Profession

It was mid-morning. I had just sat down at my laptop when an email came in about a new project. It started hopefully: there was already a clear goal, and we could use an existing process. “The requests are pretty clear,” it said. And then came that one sentence: “If you could also just briefly gather what else was needed.”

I read that sentence again. “Pretty clear... also... just...” Especially that word ‘just’ stuck with me. It sounds like analysis is something you do on the side, while you are taking notes anyway.

You Don’t Do Analysis ‘On the Side’

In many teams, analysis happens in between other things. A few questions are asked, a rough idea emerges, a wish list is made, and before you know it, you’ve started. Without deeper insights.

Until someone asks halfway through the execution:

- “What exactly do they mean by this?”
- “Is this really the most important thing?”
- “Does this need to happen now, or can it wait?”
- “Why are we actually doing this?”

If no one takes ownership of the analysis, you end up with repair work that you could have prevented beforehand. Anyone can analyze; you really don’t need a business card that says ‘Information Analyst’ for that. But someone does need to take the time and have the skills to do it right. Analysis requires attention, before and during execution. It is about staying focussed on the intention.

Can't the Project Manager Just Do That?
A role as project manager, product owner, or manager means that you steer based on value, direction, and priority of the end result. But this doesn't automatically mean that the time, knowledge, or focus is there to surface all underlying interests, dependencies, and exceptions.

That is why a good analyst makes the difference. Not to take over the role of the project manager, but to go into depth: what exactly needs to happen, why, and for whom? So yes, they can certainly do this too, but then they are also fulfilling the role of information analyst.

So in Summary, What Makes It a Profession?
It sounds easy: "just ask them what they want." But good analysis is a discipline of its own. As an analyst, you must:

- Listen without immediately thinking in solutions (and that is harder than you think).
- Dare to ask questions until it gets uncomfortable.
- Understand what people mean, even if they say something completely different.
- Dare to make conflicting interests visible.
- Translate behavior and needs into concrete actions.
- Turn vague ideas into clear, workable steps.
- Verify if what you think you understand is actually correct.

You don't learn that by doing it "just once," but by training three fundamental traits: curiosity, imagination, and communication skills.

Is Information Analysis always Necessary?

You can just start a project and see where you end up along the way. Especially in the software world, that has been the trend for years: "We're just doing it Agile."

That thought is tempting: record nothing, just start building, check together every few weeks if you're on the right track, adjust a bit, and carry on. This thought is persistent, but definitely wrong.

Projects work better with good analysis beforehand because then you know where to apply your flexibility. Being flexible is valuable and experimenting is allowed, but you do need to know *why* you are doing something, *who* it is meant for, *what* it should deliver, and *when* you do something.

Analysis is your foundation. It ensures you ask the right questions before running in the wrong direction. It might feel like a delay at the start, but it prevents costly hassle later on. A flaw in thinking discovered during a conversation at your desk costs you five minutes and a line through a drawing on your whiteboard; having to fix that same mistake only after you are finished? That costs you a multitude of wasted hours and lost trust.

Research has shown that, on average, 64% of developed ideas are rarely or never used. That is pure waste. If you manage to filter out all the fluff beforehand by asking sharp questions, you have already earned back your salary before the work even begins.

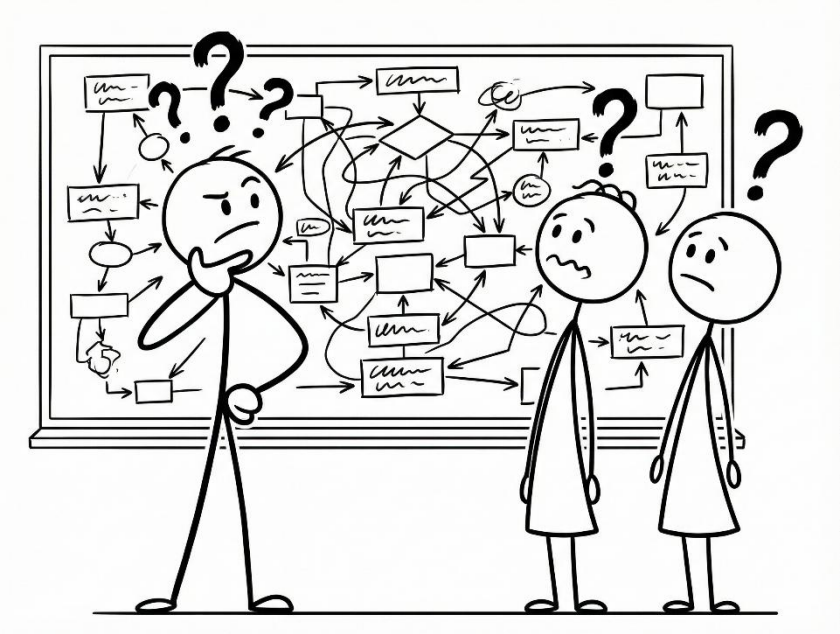
So yes, information analysis is always necessary, just not always equally extensive. Sometimes you know enough within a few hours, other times it is a full-time job.

So What Is Agile?
Agile is a way of working where you deliver value step by step, respond to change, and collaborate continuously to improve the result. It is not a method, but a mindset that revolves around:

- **Rapid partial deliveries:** You deliver work in small chunks, so you can adjust course quickly.
- **Collaboration:** You actively involve the people you are making it for.
- **Flexibility:** Plans are allowed to change if reality demands it.
- **Value at the center:** You don't work on what was 'once thought up,' but on what delivers the most right now.

As you can see: information analysis fits in perfectly there.

Information Analysis
in 3 phases



Information Analysis In 3 Phases

By now, you might be thinking: “Okay okay, I get it, information analysis is important.” There is a reason why I repeat this so often. I don’t want anyone to think: “Meh, sometimes necessary,” but rather feel: “This is essential.” If that motivates you to get better at it, my mission has succeeded.

Now, let’s dive deeper. Because without structured analysis, you unknowingly lose track of the most important things along the way:

- You lose sight of who exactly your customer is.
- The ‘why’ behind the request slowly fades away.
- The connection between the different things to do disappears.
- You simply lose your own overview.

My '<u>Line of Thinking</u>'
To prevent that, I have been working with a fixed approach for years. No complicated method and certainly no rigid model that you have to follow blindly. See it as a practical line of thinking. A common thread that prevents you from skipping something important and ensures that what you deliver actually makes sense.

It is not a checklist to tick off. It is a tool to:

- Maintain an overview.
- Make the right choices.
- Guide conversations.
- Make vague ideas concrete.

In practice, analysis rarely runs in a straight line. Sometimes you move forward, sometimes you have to go back a bit, and sometimes you suddenly realize something during the day. By having a certain structure in your head, you can make those flexible moves without losing your way.

Note: This Is Not a Waterfall Method

The phases I describe next might look like a classic 'waterfall' model on paper: first figure everything out, then detail everything, and only then build. But it isn't.

In my approach, you do work in a structured way, but you don't do everything at once and certainly not everything beforehand.

- Phase 1 (The Foundation): You lay the foundation in broad outlines at the start. After all, you need to know what you are eventually working towards.
- Phase 2 & 3 (Detailing & Doing): These run right through each other.

You work according to the principle of Just-In-Time analysis. You pick up a part of the whole, analyze it until it is sharp enough, and then pass it on directly to the team. While they build (Phase 3), you start on the analysis of the next block (Phase 2).

This way you keep speed and flexibility, but you never lose your grip.

Phase 1: From Need to Direction

Here you lay the foundation for your work. In this phase, you discover what the real question is, who is involved, and what the solution should look like in broad outlines.

You use this phase fully for new projects or for issues where the requests are still vague or abstract. For existing projects, this work is often (partly) already done; after all, without a need, no project would have started.

But watch out if you join later! Are you joining a project that is already running as an analyst? Don't just skip this phase. You always want to understand what the original need was.

During a project, assumptions often sneak in unnoticed, or it turns out that the 'why' was never really clearly defined. Dare to question those assumptions, because often things have been built on top of them. Your fresh perspective is exactly what the team needs to return to the core: what was the intention again?

The Five Core Activities
In this phase, you do five things. In this book, we will cover them one by one, but in daily practice, they often overlap:

1. **Start with an exploratory conversation:** What is the actual request? Talk to the customer to gather the first, essential information. Get a feel for where the pain or the need lies.
2. **Map out who you are doing it for:** Who are your 'customers' or stakeholders? Often there are many more than you think at first. They all have their own interests, wishes, and priorities. It is of great importance to know this early.
3. **Find the need behind the request:** What is the underlying goal? A customer often asks for a means (e.g., "I want an app"), but what do they hope to achieve with that? In this step, don't think in solutions or technology yet, but focus purely on the *why*. Why is this important?

4. **Draft it in broad outlines (preferably visually):** Make your ideas visual. Whether you describe it or draw it: ensure it is understandable. Give people something to react to. A sketch or a rough model often works better than a big pile of text. This way, you immediately test if you are heading in the right direction.
5. **Break it down into valuable chunks:** A large project is hard to oversee. Make the whole smaller by splitting it into separate building blocks that each deliver something on their own. This creates clarity and allows you to determine a rough order and priority.

Example: The Danger of "Ofcourse"

I once joined a project for an internal application halfway through. Everything seemed to be running smoothly and on schedule.

During a conversation with a user, however, someone mentioned in passing: "Yes, ofcourse the application is also meant for our external partners. After all, we view them as part of our company too."

The word 'ofcourse' got me thinking.

I looked into it, and guess what? The technical implementation hadn't accounted for external access at all. So partners would not be able to access it.

Because I caught this fairly quickly, we could adjust the design immediately. Had this only been discovered at delivery, it would have cost weeks of rework.

An Exporatory Conversation

Before you start anything, you need the basics. Are you starting a new project from scratch? Then you get that information fastest in a short conversation with the customer or the person who raised the issue.

This conversation is not yet an extensive analysis. It is an intake, meant to:

- Hear the goal and the reason in the customer's own words.
- Get a feel for the context, the history, and the people involved.
- Discover which departments or processes will likely be affected.
- Create a starting list of people you need to speak to next.

What Do You Ask in Such a Conversation?
You want to get to the core quickly. Use these questions as a guide, for example:

- **The reason:** "Why are we starting this project now? What happened?"
- **The goal:** "When is this successful for you? What needs to be there at the end?"
- **The players:** "Who has the most interest in this, or who will notice this the most?"
- **The decision-maker:** "Who ultimately gives the final sign-off on the end result?"
- **The history:** "Is there something already? Has something been investigated or built before?"
- **The boundaries:** "Are there choices that are already set in stone and that we cannot change?"
- **The risks:** "Are there sensitivities or risks I need to take into account?"

Why This Step Is Indispensable
It seems tempting to dive into the deep immediately, but this exploratory conversation saves you a lot of time later:

- You prevent a false start: You ensure that you are starting from the right problem statement.
- You learn the language: You hear which terms and examples the customer uses.
- You recognize contradictions: Because you know the original story, you notice it immediately later on if practice or other colleagues say something completely different.
- You build your network: You immediately know who you need to interview next.

This conversation is the stepping stone to the next phase, where you really start digging for details.

Tip for Ongoing Projects
Are you joining a project that is already running? Check if this conversation has already taken place. Is the foundation clear and are the agreements recorded somewhere? Great. Is it still a bit vague? Then request that half-hour with the customer anyway. It is better to check now, than to guess later.

Know Who You Are Working For

Projects where no one knows exactly who the solution is actually meant for occur more often than you think.

The customer thinks it is for their staff. The staff thinks it is for their customers. And you? You are sitting in a meeting full of opinions, assumptions, and yellow sticky notes. Everything sounds logical, but nothing aligns.
The project has probably already been approved, so there is likely a project brief lying around somewhere. That is your first source.

Sometimes you get a lot out of it; sometimes there is hardly anything on paper. But before you take even a single step, it is essential to know who you are going to need.

Therefore, always ask the question: **"Who are we actually doing this for?"**

If you don't have a clear picture of who you are working for, you won't know:

- What is truly important (and what isn't).
- What are merely side issues.
- What can stay as it is, and what absolutely needs to change.

Without a clear picture of your target group, you run the risk of building a perfect technical solution for the wrong problem.

You Work for Your Customer, But Who Is That?
When we hear the word 'customer,' we often think of the person paying the invoice, or the end user of the product. In our field, it is more nuanced. In analysis, everyone who has an interest in the result is your customer: from the end user on the work floor to the director who decides, and from the subject matter expert to the colleague who has to manage the system.

For you as an analyst, a 'customer' is anyone who:

- **Is involved** in what you are researching or making.
- **Is affected** by the result (positively or negatively).
- **Is needed** to provide you with information.

If someone is important for the success of your analysis, you treat that person as a customer. Your approach is the same for everyone: you want to understand who they are, what they need, and what they can tell you. To prevent working based on assumptions or only listening to the person with the loudest voice, use the following techniques:

1. Map Out Your Customers
Who has a stake in the result? Who gets to decide? Who depends on your work? Who needs to give permission?

To get an overview, you can map this out. A handy method is to place stakeholders on two axes:

- **Influence:** Who has the power to make decisions or steer the project?
- **Impact:** Who is affected in their daily work by the change?

This helps you focus. It prevents you from running around for the people who shout the loudest (often high influence, low impact), while forgetting the quiet key players (low influence, high impact). Precisely that last group often has the most valuable information.

2. Talk to All Groups
Speak with the people who actually do the work, not just their managers. Managers tell you how the process should go; employees tell you how it really goes.

Use the techniques we discuss later in the book in these conversations: ask about behavior, summarize what you hear, test your assumptions, and always dig for the question behind the question. This way, you not only discover what people need, but you often also uncover new stakeholders you overlooked earlier.

3. Create Personas of the Users
Sometimes the group of stakeholders is so large or diverse that you lose the overview. In that case, it helps to create personas.

A persona is a tool to make the behavior, frustrations, and needs of a group of users tangible. It is a short, recognizable description of a typical user, often with a name, a role, personal goals, and pain points. Please note: this is not a made-up marketing story, but a summary of facts you actually gathered from your conversations and observations.

A persona helps you to:

- Keep a sharp focus on what you are going to make or change.
- Understand what is valuable for that specific group.
- Gain insight into how the solution will be used later.
- Settle discussions ("We are not building this feature for you personally, but for persona 'Employee X'").

Creating personas takes time. It is especially recommended for projects involving many different roles or types of users. They help you keep a grip on the diverging interests.

Do you have a small project or are you dealing with only one or two types of users? Then an extensive persona is not necessarily needed. As long as you have a clear picture in mind of who you are dealing with and what that person needs.

You can read more about personas on page 102.

What If You Are Not Allowed to Speak to Certain People?
Sometimes you hear: "You don't have to disturb the users, just talk to me." In that case, your goal doesn't change, only your route.

Ideally, you speak with the people who will actually work with the solution. Always try to make that happen. Is that really not possible? Then you have to make do with a representative (or gatekeeper); someone who speaks on behalf of the group but is not part of it themselves.

In such a situation, you use the same interview techniques, but you are extra critical:

- **Ask lots of clarifying questions.** Don't settle for vague terms.
- **Dig deeper.** How does this person know this?
- **Check assumptions.** Is this a fact or an opinion?
- **Ask for concrete examples.** "Can you describe a situation where this happened?"

When in doubt, you can ask the representative to check your summary with the group. This way, despite the limitation, you stay as close to reality as possible.

The Need Behind the Request

This is the step before you start thinking in solutions. It is the moment where you dive into the deep to surface the real need. Not with a checklist in your hand, but with healthy curiosity and the question: "Why do they actually want this?"

People Ask for Solutions
If someone says to you: "We want a dashboard," that in itself says absolutely nothing about the goal. It only says that that person has a certain form (a dashboard) in their head because that feels logical.

Before you know it, you are busy with charts, refresh buttons, and user rights. But why? Maybe the customer wants to collaborate faster, signal errors, or just have a sense of control.

The biggest mistake in analysis is the assumption that the customer's first request is also immediately the right one. Your task is to go back to the start. *Why* that wish exists is more important than *what* exactly they are asking for.

Techniques to Find the Real Question

How do you get to the bottom of things? Not with a standard questionnaire, but by using the right conversation tools.

See the eight techniques below as your toolkit. You don't have to use them all at once. Sometimes an emotional question works, other times a drawing. Try to sense what fits.

- **Draw the work situation (or act it out):** Let someone draw or describe how a day goes. What happens exactly? Where does it get stuck and when does it go smoothly? *Drawing makes behavior concrete. You immediately see where the processes stall. This might be the most powerful technique.*
- **Ask about behavior:** Don't ask for opinions, but for facts. "When does it go wrong now?", "What are you trying to do at that moment?" and "What doesn't work then?" *People can often describe their behavior more easily than their abstract wishes. This gives you insight into the real bottleneck.*
- **The 'Why' question (without nagging):** Dig deeper. "Why is this important?", "What happens if we don't do this?" Usually, you need three to five questions to get to the core. *You peel away the layers until you reach the real value or motivation.*
- **Focus on emotions:** Ask what frustrates someone, or what makes them happy. "What discourages you in this process?" *Emotion is a very strong signal. Where there is frustration, there is often the biggest problem (and thus the most value in solving it).*
- **Sketch the future scenario:** "Imagine: three months from now, this works perfectly. What can you do then, that you can't do now?" *By letting people dream for a moment, they step out of the limitations of today and you gain insight into their goals.*
- **Ask about the past:** "What have you already tried before? Why didn't that work?" *You discover patterns and pitfalls that people are often not even aware of. It prevents you from reinventing the wheel.*
- **The Doom Scenario ("Imagine it failed"):** "Imagine, this project has failed six months from now. What went wrong?" *This reversal brings*

risks and worries to the surface that people often keep silent about in a positive conversation.

- **Summarize and verify:** Summarize what you think the other person means and check it immediately: "So if I understand correctly, you want someone to see immediately if tasks are still open without explanation?" *You remove assumptions and ensure that you are on the same page.*

You don't always have to apply all 8 techniques. Use them separately or in combination. One time emotion works, the other time a sketch, but the more often you use them, the sharper you get. You don't have to do everything perfectly, but if you really listen and dare to dig deeper via these techniques, you discover what is really going on.

Do you want to go deeper into these techniques? From page 64 onwards you will find extra explanations and examples per technique.

What Does This Bring?
If you do this well, you are not building on quicksand. You don't get vague opinions, but solid starting points:

- Goal**:** You know why we are doing this.
- Scope: You know what belongs to the solution and what doesn't.
- Alignment: Everyone understands where the journey is headed.
- Testable: You can verify at any time if your solutions match the need.
- Prioritization: You know what is a main issue and what is a side issue.

And not unimportantly: it makes your work more enjoyable. Building a solution that you know is really needed gives much more satisfaction than guessing what the customer wants.
Tip: Write down everything you hear**.** You don't have to write a novel, but do note down what you hear. Often, precisely that one sentence you now think is: "Oh, that's just a detail," turns out to be the key to the solution later.

Designing (Visually) in Broad Outlines

Information analysis is about one thing: understanding what is really going on. You can't achieve that with words alone, nor with pictures alone. You need both.

At the same time, you want to avoid producing a massive document. The more text, the greater the chance that no one will read it. Therefore, stick to the rule: **if you can draw it, draw it.**

Especially in this phase of the project, where it's not yet about the details but about understanding and direction, a sketch works many times faster than a wall of text. A process, a screen, or an information flow can be drawn perfectly. By making it visual, you see exactly where it works and where it doesn't.

Does this mean you have to draw everything? No. There are statements that you actually need to capture literally in words, because they reveal the core of the problem. Think of sentences like:

- "I always double-check it in Excel, because I don't trust the outcome."
- "I don't dare press that button because last time I lost everything."
- "I have to fill in so much that it feels unnecessarily complex."

These kinds of sentences are valuable. Write them down literally and incorporate them into your analysis.

The Effect of Drawing: Eliminate Interpretation

Our brains are wired visually. A simple drawing on a whiteboard or notepad gets people thinking immediately. They suddenly see what they previously only heard.

Working visually is not an extra step; it is an accelerator. As long as something exists only in words, there is room for interpretation. Everyone creates their own picture in their head. But as soon as you look at a sketch together, that noise disappears and people get sharp:

- "Hey, this isn't right."
- "There is a step missing here."
- "Is this really what you mean?"

Example: "I don't think that is what I meant"

I once explained a solution we had come up with. On paper, everything seemed right: I used the customer's words, the requirements were neatly phrased, and a colleague had checked it. Yet, I saw confusion in the customer's eyes, and he said: "I don't think that's what I meant…"

I grabbed a marker and made a quick sketch on the whiteboard: this is what you see, you click here, and then this and that happens…

He nodded immediately. "Ah, yes, *that* is indeed what I meant."

What Do You Draw (And What Don't You)?
You don't have to be a designer. In fact: you don't have to be able to draw beautifully at all. Visual design here is not an art form, but a communication tool. You are not drawing for a system, but to understand each other.

Types of drawings that often work well:

- **Screen sketches (wireframes):** What is where and what does the user do?
- **Flowcharts:** When does what happen? In what order? What is manual and what is automatic?
- **Who-does-what:** Who does what, when, and who is waiting for whom?
- **Information flows:** Where does the information come from and where does it go?
- **Before-and-after:** How does it work now, and what will the world look like later?

Drawing Leads to Better Questions
While drawing, you force yourself to think logically. You automatically come up with questions you might otherwise have forgotten:

- "Is this step actually needed?"
- "What happens if this isn't filled in?"
- "Who actually gets to see this?"

It helps you to think beyond "sounds good" and makes implicit assumptions painfully visible.

Another example: The approval

The request is simple: “We want to make it easier to approve a request.” If you ask what that means, you often get vague answers. But if you draw it:

📄 Document → 🖱️ Click button → ✅ Approved → 📤 Send

...then someone suddenly says: “Yes, but they do have to accept the terms and conditions first.”

And suddenly you are no longer talking about assumptions, but about concrete behavior and rules.

Tools? A Single Sheet Of Paper Is Enough

Digital tools are useful, but don't let that stop you if you don't have a laptop at hand. Draw on paper, a whiteboard, a napkin, or a coaster. Take a photo of it and paste it into your documentation.

Finally: A Stepping Stone to the Next Phase

Working visually has another big advantage: it helps you break down your work. As soon as you draw out a process or screen, you often automatically see where the logical ‘cut’ lies. Every box or every step in your drawing can be the start of a separate building block that you will analyze further.

Breaking It Down and Prioritizing

If all goes well, the previous steps have given you a clear picture of the whole. You understand the request, the boundaries are clear, and together you have sketched what the solution should roughly look like.

You don't know all the details yet (and you don't have to), but you've got the big picture. However, you don't build that entire picture all at once. Therefore, the next step is: breaking it down. You cut the large project into logical, functional 'building blocks.'

For the IT Professionals Among Us
In this book, I use the term *building blocks* because it is understandable for everyone. Do you work in software development? Then you probably recognize these as User Stories or Features. The principle is exactly the same: you make the work small and manageable.

Why Break It Down?
A large goal is vague. Small parts are concrete. The smaller you make it, the better you can:

- Estimate: How much time will this specific piece take?
- Prioritize: What is truly important now and what can wait?
- Plan: Who picks up which part?

Compare it to building a house. Through the earlier steps, you know what the house looks like and who you are building it for. You could say: "The house will be finished in a year." But how do you determine that? You have to break it down into tasks:

- Pouring the foundation
- Laying the brick walls
- Installing the electrics
- Placing the roof

When you see these separate parts you understand the dependencies (first the walls, then the roof) and can you estimate what is involved. That's how it works with your project as well: break it down first, then execute.

Note: No Details Yet
In this phase, we are still working in broad outlines. You make it workable, but you aren't going to detail every step or every exception yet. For each block, you primarily describe:

- What needs to happen?
- What should the result be?
- Who is it intended for?

Details like "What color is a button?" or "Is there an exception?" come later. The goal is to cut the work into understandable blocks, ensure you don't overlook anything important, and create pieces of work that you can refine, estimate, and prioritize.

Design From Behaviour, Not From the Solution
A common mistake is breaking things down into solutions, such as "Build a menu" or "Create a dashboard." That says nothing about the value for the user.

Therefore, always break it down based on **behavior**. What does someone want to be able to do? Take a customer portal, for example. The request is: "We want a self-service portal for customers." You don't break this down into screens, but into actions:

- Customer can change contact details.
- Customer can view outstanding invoices.
- Customer can submit a request.
- Customer can view the status of an application.

Each block now represents value for the user. Only when detailing each block do you look at exactly how the solution will work.

A Structure That Always Works
To ensure your building blocks stay concrete, you can use this standard sentence (well-known to IT professionals):

As a [role]**, I want** [action]**, so that** [goal]**.**

For example: "**As a** doctor's desk clerk, **I want** to see immediately on the first screen who a customer has an appointment with, **so that** I can direct them to the correct consultation room upon arrival."

This is clear, measurable, and workable. Do watch out for "fake goals," though. Ensure the part after 'so that' is actually a goal.

Wrong: "...so that I can do my work." (This says nothing).
Wrong: "...so that I can enter it into the system." (That is an action, not a goal).

Prioritizing With MoSCoW
Now that you have a list of building blocks, you will notice you have more work than time. You have to choose. A handy method for this is MoSCoW. Preferably you do this together with your customers:

- **M - Must have:** This must be in. Without these parts, the solution won't work or the project fails.
- **S - Should have:** Very important, but if absolutely necessary, it can come a bit later or via a (temporary) workaround.
- **C - Could have:** Nice extras ('nice to have'). We only do this if there is time and budget left.
- **W - Won't have:** We aren't doing this now. Maybe in the future; we'll decide that later.

This method ensures that you and your customers make clear choices. It prevents endless discussions where everything is called ‘important.’

You can read more about MoSCoW on page 99.

It Doesn't Have to Be Perfect (And Order Is Something Else)
The first list of building blocks doesn't have to be perfect. In fact, you won't manage that anyway. During the project, you’ll naturally discover that some blocks are too large (you'll split those) or that priorities shift. That’s part of the process. Ensure you have a logical starting list, and you’re already halfway there.

Important difference: Priority vs. Order
MoSCoW determines what is important (the value). That is not always the same as the order in which you build. Not all ‘Must haves’ need to be finished first. Sometimes you start with something small to gain trust, or with the most difficult piece of technology to eliminate a risk. You determine this order (the schedule) in the next step.

Phase 2: Detailing the Specifics

In Phase 1, you worked on the big picture. You know who you are working for, what the core question is, and you have broken down the solution into logical chunks with a clear priority.

Yet, you aren't there just yet. A rough idea is not yet a workable plan. You cannot simply tell a team: "Just build this part," without further explanation. To move from a plan to execution, you need to zoom in.

From now on, you will go into depth for each of your building blocks. You are making the translation from an abstract idea to concrete actions. The goal of this phase is simple: ensuring that everyone knows exactly what needs to happen. Also include the *why* here, regardless of how big or small your building block is. When people understand why something needs to be done, they can actively contribute their ideas.

Detailing the Building Blocks

You might recognize this: the entire group is ready to make decisions, and then someone says: "Well, we actually need to look into this a bit more first."

The result? Thirty minutes of discussion with questions like:

- "What exactly do they mean by this?"
- "Is this already final?"
- "How does this work technically?"

The conversation stalls, assumptions are made, and eventually, everyone leaves without a real decision. A waste of time and completely unnecessary.

The solution: **preparation.**
You prevent this by ensuring a building block is already researched before it enters the group discussion. That is your task in this phase. You provide the necessary clarity beforehand, so the team can make quick and solid decisions later.

Your approach this the same way as in the large-scale analysis as explained in phase 1, but on a smaller scale. You zoom in. Your activities are:

- **Deepening the content:** What exactly is the core of the request or the problem? (The *why*)
- **Holding conversations:** Talk to the people who are directly involved with this specific piece of work.
- **Describing behavior:** What exactly does the user do here? Which actions do they perform?
- **Looking for exceptions:** What happens in 'difficult' situations or when something goes wrong?
- **Checking rules:** Are there laws or internal agreements we need to take into account?
- **Sketching:** Make it visual; what should it look like?

Your goal is clarity. No more vague ideas, but a sharply detailed piece of work that the builders can start on immediately.

Example: The shower

Earlier, we compared the project to building a house. We then divided the house into rooms (the building blocks). Now, you zoom in on one specific part: the shower.

To detail this building block, you ask targeted questions again. No longer about the entire house, but purely about this specific detail:

- Where exactly in the room will the shower be placed?
- Do we want a rainfall showerhead or a handheld one?
- Will it be a walk-in shower or a cabin?
- Does the floor need to be non-slip?

The more concrete you make this, the smaller the chance that the contractor will build something you didn't want later.

What Should It Deliver?
A well-defined building block meets these requirements:

- Understandable: Everyone gets what is written down.
- Realistic: It aligns with what people actually do.
- Concrete: Clear choices can be made.
- Discussion-worthy: Scenarios are ready to be brainstormed or debated.
- Validated: The people who actually have to work with it have already reviewed it.

Tip: Create a Standard Template
Use a fixed layout for every building block. This way, you won't forget anything, and it becomes recognizable for everyone. Make sure it includes (at least):

- Why this is needed.
- What it should deliver (the goal).
- How it works (the content description and rules).
- What it looks like (sketches).
- Risks and assumptions that haven't been tested yet.
- Dependencies (do we need something from another team?).

How Do You Approach This?
How do you do this efficiently without drowning in details? Use this workflow:

1. **Start with the highest value**: Pick up the building blocks with the highest priority first. These are the parts that deliver the most value to the user. By starting here, you ensure that the team (the executors) can immediately get to work on the most important tasks. *Note: You must stay ahead of the pack. Prevent executors from sitting idle because you are still analyzing. As soon as a block is clear enough, hand it over and start on the next one.*

2. **Verify the order:** Look critically at whether the sequence you previously came up with is still logical. Discuss this with the executors as well. Ask them: "What is a logical starting point for you?" Sometimes it is technically more convenient to start with block B before block A is built.

3. **Use your interview techniques (but zoomed in):** When detailing, use the exact same conversation techniques as in Phase 1, such as asking for the 'need behind the request.' The difference is the scope: your lens is now zoomed in.

> You no longer ask: "What is the goal of the system?" You now ask: "Which data do you need here?" and "What should happen if things go wrong?"

The sharper your vision becomes, the smaller the chance of surprises and errors during execution.

The Litmus Test: INVEST
A handy tool to check if your building block is truly ready to be picked up is the INVEST checklist. Measure your work against this list:

- **Independent**: The component stands on its own as much as possible. It isn't tied to all sorts of other tasks, allowing you to build and test it separately.
- **Negotiable**: It is not an ironclad contract. There is room for the team to contribute ideas for the smartest technical solution.
- **Valuable**: It delivers something immediately for the user. If you can't explain why this adds value, you need to go back to the drawing board.
- **Estimable**: It is clear enough to say how much time it will take. If the team says: "No idea how long this will take," your description is probably still too vague.
- **Small**: It is manageable. Is it too big? Then break it down into smaller pieces. This way, you maintain speed and see results faster.
- **Testable**: You can objectively determine when it is ‘done.’ There are clear criteria that the end result must meet.

Do you see gaps when you go through this list? Correct them immediately. This prevents you from getting stuck during later group sessions.

Read more about INVEST on page 105.

The Joint Review

Analysis is not something you do on your own. As an analyst, you have already done a lot of prep work: you have held conversations, gathered information, researched the facts, and worked out a concrete proposal. You've likely already bounced ideas off your colleagues to sharpen your thoughts. But before you truly get started, you need to be sure about one thing: is what's on the table correct? And perhaps even more importantly: is it actually possible?

That is why you always discuss your proposal with the people who will execute the project. They help you check whether the idea is feasible in practice. They point out details you might have missed and often come up with clever solutions you hadn't thought of yourself.

The implementers are often the first to see if something truly works. Perhaps there is a (technical) constraint, or your proposal is much more complicated to implement than it seems. You want to discover these things before the work begins, not halfway through. Do adjustments turn out to be necessary after the conversation? Then you process them immediately or provide feedback. This keeps everyone aligned and prevents surprises later on.

What happens during such a session? It is not a one-way street, but an exchange:

- **You present the content:** What needs to happen, who are we doing this for, and why is it necessary?
- **Colleagues verify the content:** Is the story correct, both functionally and technically?
- **Specialists add their expertise:** They contribute their professional knowledge on points you might not be aware of.
- **Critical questions are asked:** Together, you look for blind spots and things that have been forgotten.

- **The order is discussed:** What makes sense to do first and what comes later?
- **Constraints become clear:** Are there boundaries or technical limits we need to take into account?
- **Brainstorming for solutions:** How can we tackle this in the smartest way?
- **Work estimation:** How much time and effort will this take?
- **Decision-making:** The decision-maker indicates whether this is feasible and valuable enough.
- **The final conclusion:** Do we know enough to get started?

If you do this well, it's not a boring meeting but an energetic working session. It is the crucial step to prevent errors and start with confidence.

Example: Everything was right, until we looked at it together

The analyst had done his homework thoroughly. The customer had a clear request: employees needed to be able to upload their own profile pictures for their travel passes. This photo would then be placed on the pass automatically.

The goal was clear: less administrative work for HR, a faster process, and employees having control themselves. The analyst had thought of everything:

- Spoke with HR and the end users.
- Mapped out the different types of users (personas).
- Created a sketch of the screen and the process.
- Thought about exceptions (what if the photo is the wrong format?).
- Wrote down the requirements it had to meet (acceptance criteria).
- Aligned everything and got it approved by the customer.

In short: on paper, everything seemed ready to go. But when the team came together to review it all, new questions emerged from different perspectives:

- The tester asked: “What happens if someone uploads an inappropriate photo as a joke? Who checks that?”
- The builder asked: “Where are we actually storing that photo? In the HR system? Or somewhere else? Because a link with the HR system needs time.”
- The privacy specialist noted: “Careful, we cannot just store this personal data without explicit consent from the employee.”

Through this conversation, it became clear that the plan was not yet complete:

- An extra step was needed to verify the photo before it was placed on the pass.
- The storage location needed technical alignment.
- A clear notification had to be added where the employee gives consent for the use of the photo.

The result
An extra building block was added for the photo check, the analysis for the storage was revisited, and requirements regarding privacy were added. It cost fifteen minutes of extra discussion, but it prevented weeks of rework later on.

Your Role as a Content Guide
You are not at the table simply to read your document out loud or to tell everyone what to do. You are there to ensure the story is correct and that everyone understands it. You facilitate the conversation.

This means that you:

- Invite people to voice their questions and doubts.
- Ask probing questions, such as: “What are we still missing?” or “Is this component essential?”
- Honestly state what you haven't researched yet and discuss whether that is a problem.
- Keep checking: “Do we all truly understand this in the same way?”

How to Get it Right (6 Tips)

1. **Start with behavior, not solutions:** Don't read your work from A to Z. Start with the goal and the behavior. Say, for example: “The user wants to achieve goal X in situation Y.” Don’t say: “I want a button here and an input field there.” By starting with the behavior, you invite specialists to think along about the smartest solution. Maybe that button isn't necessary at all.
2. **Use visuals:** A picture is worth a thousand words. Use the sketches you’ve made or draw on a whiteboard during the conversation. Flowcharts or simple drawings turn an abstract story into something concrete immediately. Show what you mean.
3. **Leave room for surprise:** Don't fall in love with your own solution. Perhaps a colleague has come up with a much smarter or faster approach than you. Actively ask for it and give space for that creativity.
4. **Lay the risks on the table:** Don't pretend everything is certain. Are there dependencies on other teams? Is certain data perhaps unavailable? External factors that have an influence? Name them now so you don't trip over them later.

5. **Talk about testability:** When is it ‘done’? Discuss how you will check if it’s successful later. What exactly needs to work or be measurable?
6. **Determine the order together:** In this session, you determine the sequence in which the work will be picked up. You probably already have an idea (for example, via the MoSCoW method), but this is the moment to test it.

   The order depends on four factors: **priority**, **depencencies**, **risks**, or just a **quick win** (sometimes I just want to pick something small for quick results.)

When a building block is accepted by the executors they can get to work immediately, while you work ahead on the next blocks.

When is Something Truly ‘Ready to Do’?
At the end of the session, ask the question: "Do we know enough to start?" The answer is only ‘yes’ if:

- Everyone understands what the intention is.
- The requirements are clear (we know what ‘good enough’ looks like).
- The approach and workflow are clear.
- Deviations and edge cases have been discussed.
- There is an estimation of the amount of work.
- There is agreement on the order of work.

Only when everyone gets it can someone actually build it.

Phase 3: Execution and Adjustment

Now it's actually happening: plans are becoming reality. The team gets to work on the agreements that were made. For all building blocks (yes, even the small ones) you remain essential. Questions you didn't foresee will always arise. By being available, you prevent small misunderstandings from snowballing into major errors or incorrect assumptions.

Regularly check whether the result still aligns with the original intent. Ask the people you are doing the work for. In practice, new insights often emerge that you could never have come up with from behind a desk.

Don't be afraid to adjust things. Whether it's a minor detail or a major change: discuss it openly with your team. Sometimes this means priorities shift. That's fine, as long as it ultimately leads to a solution that works.

From Analysis to Reality

The start has been made. The agreements are set, everyone knows what needs to happen, and the course is clear. You might be leaning back now, thinking: "My work is done, good luck with it."

Unfortunately, that is rarely the case. In fact, it is right now that your analysis is truly being tested. No longer on paper or in theory, but in the gritty reality of practice.

Why Your Work Doesn't Stop Here
The moment people actually start working with your analysis, the nature of the questions they ask changes. In the preparation phase, it was about the broad strokes; now, it's about the specifics.

You can never make everything 100% bulletproof beforehand, and that's okay. But prepare yourself for questions like:

- "What should the system do if this input field remains empty?"
- "What do we do if a zip code is missing from the data?"

If you aren't there to provide an answer at that moment, something risky happens: people start guessing. Someone makes their own interpretation or postpones the decision. The result is a solution that is 90% correct but misses the mark on the crucial details. That is why you stay available. Not to play police officer, but to guard the original vision.

What Do You Actually Do During Execution?
Your role shifts from 'creator' to 'guide.' These are your most important tasks:

- **Answering team questions:** Ensure you are available for content-related questions. At this stage, you make the difference between doubt and clarity. Don't know the answer yourself? Check with the customer. The team can sometimes do this themselves, but you know the background and the goal best, allowing you to get to the core faster.
- **Guarding the intent behind the solution:** During build or execution, people often come up with new, clever ideas. That's great and should be encouraged. Your task is to test whether that 'smart approach' still fits the original goal. Does it still solve the problem? If so: go for it.
- **Spotting deviations and shortcuts:** Sometimes people choose the easy way out because they are in a hurry. That isn't always bad, but you are there to assess if such a 'shortcut' is acceptable. Does this still align with our agreements, or are we sacrificing too much quality?
- **Recognizing when the approach no longer works:** Sometimes you only see during execution that what you came up with doesn't actually work well in practice. Dare to hit the brakes. Ask the question: "Should we stubbornly push through, or do we need to adjust the plan?"

Your Analysis Is Not Sacred
I mentioned it before, but a major pitfall is falling in love with your own ideas and designs. Do not treat your analysis as the Holy Grail that cannot be deviated from. It is the foundation, but practice is the ultimate test.

Are corrections needed? Fine. Always do so based on the core questions you asked at the start: "What is this supposed to solve?" and "What were we trying to achieve with this?" As long as the answer to those remains solid, the solution is allowed to change.

Gathering Feedback

Your solution is finished, tested, and handed over. Perhaps people are already working with it. And then comes that inevitable moment. During a presentation or in an email, you hear: "It looks good… but now that I actually see it, I'd like it to be just a little different..."

That is frustrating. You did your entire analysis right and thought you were done. Yet, this is often not due to ill will, but a sign that feedback was gathered too late. The rule in our profession is simple: the later the feedback arrives, the more time, energy, and money it costs to adjust something.

Why People Speak Up Late
People often wait until the very last moment to offer their comments. They usually don't do this to be annoying, but because of one of the following reasons:

- They don't want to 'nag' or seem ungrateful: People are often just happy that hard work has been done and there is a result. They keep their doubts to themselves to keep the peace or because they are afraid their remark will be taken as personal criticism.
- They assume you are the expert: The thought is often: "The specialists must have thought this through, so who am I to question it?" Because of this, illogical steps in a process are sometimes unintentionally accepted.

- They only notice it in practice: A design on paper or a description in a document looks very different from reality. Only when someone actually starts working with the solution do they feel where it works and where it doesn't.
- They can't explain their problem well: Sometimes people just have a gut feeling that something isn't right, but they don't know exactly what the solution is. Because they can't propose a concrete improvement, they say nothing at all upfront.

It is therefore your job to actively ask for feedback. Don't wait for them to come to you, because by then, you are usually too late.

When and How Do You Gather Feedback?
The short answer: as soon as possible. Absolutely do not wait until everything is 'finished.' As soon as there is something people can see, feel, or experience, you can ask for feedback. Formats you can use include:

- **A prototype or sketch**: A simple visual representation of a screen or process, even if nothing actually works yet.
- **A partially working solution**: A small component that already functions, so that people can test the basics.
- **A simulation**: Acting out the new work process, perhaps just on paper or through role-playing. It's a good idea to do this earlier during the detailing phase, but it's never too late.

The keywords here are "experience it". Someone who tries something themselves provides much richer feedback than someone who only passively watches a presentation or reads a text.

How to Get Good Feedback?
People tend to give socially desirable answers. To hear what they actually think, use these three techniques:

1. **Let people do it themselves:** Give them the space to use the solution. Don't sit next to them to explain how it works; instead, observe in silence.

   What people do often says more than what they say. If you see someone looking in the wrong place three times, that is valuable information, even if that person says afterward: "Oh, it was all very clear."

2. **Ask open-ended questions:** Ensure it feels safe to be critical. Avoid closed questions or ones like "Do you get it?" (no one wants to seem slow, so everyone says "yes"). Instead, ask questions that invite them to tell a story:
   - "What would you do at this point if I weren't here?"
   - "What did you notice while using it?"
   - "What worked differently than you expected?"
3. **Take it seriously, but promise nothing:** Show that you appreciate the critical points. Write everything down while they are watching. However, be clear in your expectation management: you are noting everything to investigate it, but you cannot promise that everything will be adjusted immediately.

What Do You Do With the Feedback?
Don't rush back to your desk to overhaul everything the moment you receive a comment. First, analyze the feedback calmly based on the following points:

- **What is the core problem?** Try to look past the immediate remark. If someone says, "I want this button to be red," the problem might not be the color, but that the button doesn't stand out enough.

- **Is it isolated or structural?** Is only one person struggling with this, or does everyone run into it?
- **Is it a blocker or a wish?** Is it impossible to continue working (a 'showstopper'), or is it merely a “nice-to-have” request?
- **What is the urgency?** Does this need to be solved now, can it wait until a next version, or will we do nothing with it?

Discuss this analysis with your team of executors and your customer. Together, you determine the next step: adjust immediately, go back to the drawing board, put it on the list for later, or consciously do nothing. The latter is also a valid choice: sometimes the current solution is simply good enough.

Aftercare is Also Part of the Process

The solution has been delivered and is working as planned. Time to celebrate! You might be thinking: “Great, now my work is truly done.” But no, not quite yet.

The aftercare phase is necessary because this is the moment of truth. Now it becomes clear whether everything you’ve conceived, aligned, and tested actually works consistently in daily practice. No matter how well you’ve organized the hand-over, you are often the one who best understands why it works the way it does, and therefore also where the weak spots are.

Your Role Changes, But Doesn’t Disappear
It’s not just about the solution working now; it’s about it continuing to align with the way people work. A good analyst remains involved in this phase. Not to “play helpdesk,” but for three specific reasons:

- You know the history: You know exactly why certain choices were made in the past.
- You are the bridge: You understand the gap between the theory (the plan) and the practice.

- You recognize deviations: If something goes wrong, you are quicker to see if it's an isolated incident or a structural flaw.

The Questions Heading Your Way
In this phase, you often act as a knowledge hub for users and administrators. You assist with questions like:

- "Is this a bug, or is it supposed to work this way?"
- "Why is the system responding differently than we're used to in the old one?"
- "Can we still make this tiny adjustment?"
- "How exactly did this process work again?"

Watch for Silent Signals
In addition to answering questions, you also observe in this phase. Sometimes people say nothing, but you can see from the data or behavior that something isn't running smoothly.

Be alert for:

- Anomaly data: Data that looks strange or illogical is often a sign that people are using the system in an unforeseen way (which shouldn't be possible, so an adjustment is desirable) or that something is simply going wrong somewhere.
- Creative workarounds: Users who "officially" do everything right, but in practice use sticky notes, Excel sheets, or other tricks to work around the system.

Identifying this type of behavior isn't about policing; it's pure information analysis. It tells you exactly where the solution does not yet seamlessly connect with reality.

A Detailed Example

Up to this point, you have mostly read about the approach. Great words, clear steps. But if you're anything like me, you know that one practical example often says more than ten pages of theory. So: how does it really go, from a vague idea to something that works?

Analysis isn't a matter of reading a book and being done with it. You have to put in the miles. Gaining experience, digging deeper, aligning, detailing, starting over. Sometimes you're spot on; sometimes you miss the mark. That's all part of it.

The beauty of the method in this book is that you can adjust at any moment. You discover things while detailing, your team sees where attention is needed during execution, and your customer notices quickly what does and doesn't work.

So, what if you aren't completely thorough right away? No problem. As long as you started with the right things. That leaves room to improve, expand, or adjust.

That being said, below is a fully detailed example of a small IT project, from idea to concrete result, including moments where course corrections were made. Even if you don't work in IT: read it anyway. The approach you see here applies to almost any project, IT or otherwise.

Phase 1: From Need to Direction
A team leader from the field service department approaches you. "I would like a nice new planning tool for team leaders so that we can properly schedule the work for our teams."

That sounds like a clear assignment. But you sense it immediately: there's more to this. So, you start digging deeper:

- "What exactly is the problem you are experiencing right now?"
- "Who else is looking at this planning?"

You soon discover how things are currently handled: they work in a 'shared' Excel file. In theory. In practice, there are many copies floating around. The result is that it isn't always clear what others are doing, employees are double-booked, appointments are being missed and errors creep into the invoicing.

The real need isn't a "nice tool for team leaders," but overview and alignment for everyone involved. Less duplicate work, less chaos. The team leader said it was for only for team leaders but you are wondering about other people in the whole planning process.

Who do you need?
You map out who you need to speak with. You keep asking: "Who else does this affect?" This creates the following picture:

- Team Leaders: Want to plan and adjust easily.
- Employees: Simply want to see on their phones where they need to be tomorrow.
- The Manager: Wants to see reports on productivity.
- Administration: Wants to easily import the hours into their invoicing system.

You speak with people from all these groups, check if they recognize the problems and needs, and create sketches to visualize your ideas (a process drawing, sketches of a weekly overview for the planner, and a mobile screen for the employee).

After a few conversations, the picture is clear: you know what they want, why they want it, and what the solution should roughly provide. Armed with a head full of ideas and your notes, you head back to your desk.

Breaking it down into building blocks
You define four logical building blocks and determine the priority (MoSCoW):

1. Must Have: As a team leader, I want to plan activities for my team, so that I can deploy them efficiently.
2. Must Have: As an employee, I want to view my daily and weekly schedule, so that I can prepare for my work.
3. Should Have: As a manager, I want to view reports, so that I can manage deployment.
4. Could Have: As administration, I want to export data, so that I can invoice faster.

The focus for the first version is on the team leaders and the employees. That's where the biggest pain is. The rest will come later.

Phase 2: Filling in the Details
Now you zoom in on the most important building blocks and work them out further.

Building Block 1: As a team leader, I want to plan activities.

- Details: An activity has a title, start time, end time, work type, employee, and a location.
- Rule: A notification appears if more than 8 hours of work are planned for a day.
- Rule: A warning follows a double booking (but it doesn't block it, as overtime occurs).

Building Block 2: As an employee, I want to see my schedule.

- Details: Show activities per day.
- Details: Show activities per week.
- Rule: The employee must be able to add notes per activity.

The building blocks above are not a fully detailed example; in practice, you would add more details such as design sketches, logic, and exceptions.

Based on these blocks, you conclude that the team leaders also need an overview of the total schedule and must be able to filter this overview by team. So, you write an extra building block for this.

Once you're ready, discuss the details with the customer's user groups first. Together, you look at it: is everything correct? Is anything missing? Is the intent clear? When you no longer see major gaps or ambiguities, you conclude: this is solid enough to start.

The users agree, so time for the next step: the development team. They haven't been sitting idle, by the way. While you were busy with the analysis, they already started laying the technical foundation based on the initial assignment.

During the conversation with the team, you go through the building blocks. Questions and suggestions arise:

- "If we make filtering by team possible, filtering by employee is also easy to add."
- "Wouldn't it be handy to have a 'copy appointment' button?"

Great, a team that thinks along! Neither you nor the customer had thought of this yet. You include the filter function immediately (little work), and you put the copy function on the list as a building block for later (nice to have).

Because you prepared well, there is room for these kinds of improvements. Everyone can get to work.

Phase 3: Execution and Adjustment

The development team starts. You aren't sitting still but are already working on the analysis for the next blocks. In between, you answer questions from the team:

A question comes in: “What do we do if someone plans something on a national holiday?”
That situation wasn’t in your design. You consult: it is allowed, but it must be visible. So, you add it.

After two weeks, there is a simple, partially working version. You have a team leader and an employee test it. They respond:

- “It’s clear; I understand it immediately.”
- “But… I really want a ‘copy appointment’ button. I often plan the same thing.”
- “I can't see who changed my schedule. I find that annoying.”

New insights emerge. Not because anything is wrong, but because usage always yields new insights. You already had the copy function because the development team had suggested it. It turns out to be high on the customer’s wish list. As a result, the building block moves up from "build later" to "build right now." The development team includes it in the planning immediately.

You also create a new building block:

As a team leader, I want to be able to see who adjusted the planning, so that I can trace changes in case of errors.

You discuss the details of this new request with your customer and the team and determine the priority. You conclude that this request is more important than reporting for management and administration, so this block is also discussed and included in development.

The Result
Two weeks later, the first version is ready. Did we build everything people shouted for at the beginning? No. The manager and administration still have to wait, and the system isn't very refined yet. But what is there

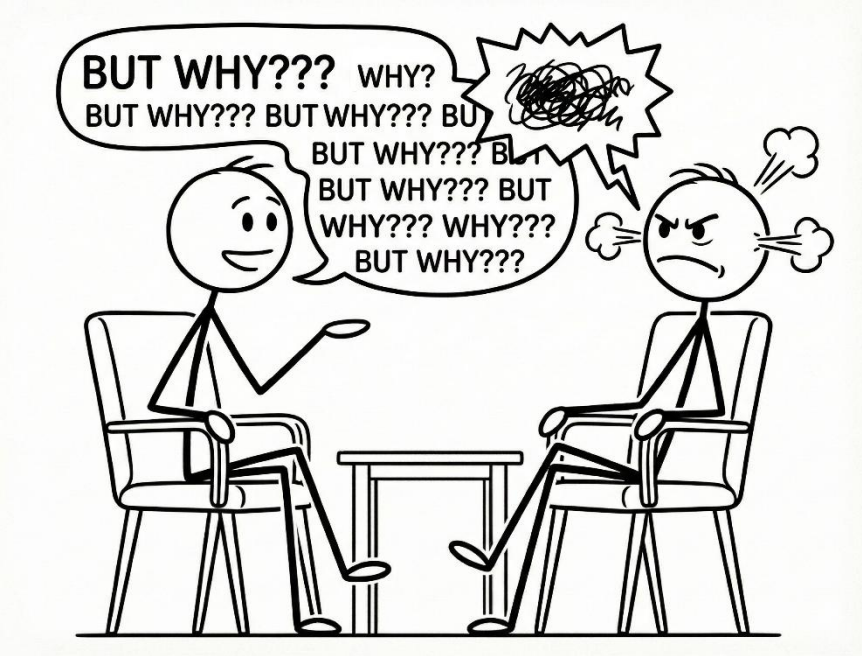
solves the core problem: team leaders can plan without errors, and employees know where they stand. The chaos is gone.
Now that the foundation is set and the users are satisfied, you start the second round for the manager, followed by administration. And when that's done? Onward to improvements in the existing system and the building blocks that were given low priority.

You didn't just build 'a tool.' You delivered a solution that fits the reality of practice. Something that truly makes people happy and makes their work better and easier. And that is what good analysis is all about.

What You Should Remember:

- **Never start building immediately:** The first request is rarely the real problem. Keep asking questions.
- **Involve all groups:** Constantly ask: Who uses this? Who needs to have an opinion on this?
- **Work in building blocks:** Break it down and prioritize what delivers the most value right now.
- **Fill in details later:** Only work out details when a block is up next, and test them with users.
- **The Joint Review:** Involve the development team early; that's often where the best ideas come from.
- **Stay available:** Many insights only emerge during building and testing, and even after delivery.
- **Embrace feedback:** Use new insights to improve and dare to shift priorities.

Conversation techniques



Conversation Techniques

You can follow the most rigid models and use the smartest tools. You can work with templates until you drop. But without a good conversation, you get nowhere.

As an analyst, your most important instrument is not your laptop, a model, or a system. Of course, those things help you structure your work, but you extract the real content from a conversation. A conversation in which you probe effectively, filter out the noise, and get the actual need out in the open.

In this chapter, I will provide you with some of the techniques to hold such conversations. Do not expect standard checklists that you tick off like a robot. I am giving you techniques that truly help you get to the core. Techniques that help you uncover the wishes, concerns, and assumptions of your customers, even if they don't have them clearly in mind yet.

Drawing

Drawing isn't exactly a conversation technique, but it is one of the most powerful things you can do during a meeting. That is why I want to cover this topic before we move on to actual verbal techniques.

Sometimes (actually often) words alone won't cut it. Someone explains something, but it just doesn't click. Or the explanation remains vague: “Yeah, and then it gets stuck during the validation,” or “That’s where the process gets a bit unclear.” What do they mean exactly? Where does it go wrong? How does it work? In these cases, there is one solution that almost always works: have them draw it.

Drawing is ‘thinking with a pen.’ By making a situation visual, you turn the abstract into something concrete. It forces a person to think logically about the sequence and the content.

For you as an analyst, it immediately becomes clear where the problem lies. Assumptions surface ("Oh, do we always do it that way?") and you see exactly where departments, systems, or people clash. It doesn't have to be a masterpiece; a simple sketch on a flip chart or a piece of paper often says more than twenty questions.

How Do You Do It?
You don't have to be an artist. Neither does your conversation partner. Grab a marker and paper (or walk to the whiteboard) and start.

Questions that help with this:

- "Could you draw how that works currently?"
- "Can you sketch out what you do, when, and with whom?"
- "Could you put an 'X' exactly where it goes wrong?"

Choose a format that fits the topic:

- **A Flowchart:** The format you can use for almost anything. The beauty of this is that it makes every 'if-then' scenario easy to visualize. *(See page 92 for more on this type of drawing).*
- **A Timeline:** For processes that happen over time.
- **A Block Diagram:** For the route through departments or systems.
- **Stick Figures:** To show who is working with whom.
- **Whatever Works:** If you have no idea which format fits, combine them. It's about the outcome.

Tip: Allow for silences while someone is drawing. This is often the moment they discover for themselves how their process actually works or where it falls apart.

Alternative: Acting It Out
Some people don't like to draw or don't think visually. No problem. Have them 'act it out' or tell it step-by-step:

- "What is the first thing you do?"
- "And what does your colleague do then?"
- "And what do you do after that?" Meanwhile, you draw along on paper to make it visual for them anyway.

When to Use This?
As often as possible, but especially when the conversation gets stuck in vague terms like "Just how it always goes" or "It breaks down between department X and Y." Especially at handover moments (between people or systems), drawing is indispensable for finding the gaps in the story.

Example: The vanished E-mail

An employee mentions that deliveries are sometimes delayed between the warehouse and transport. You ask: "Can you draw that for me?"

She sketches: Production → Warehouse → Transport.
At the arrow toward transport, she writes: "Via email."

You point to the arrow: "And what happens if Transport misses that email?"
The answer: "Well, then the delivery just sits there."

Through that simple drawing, everyone saw the problem: the process relied on an unreliable email. The solution wasn't 'paying more attention,' but a system that shows an outstanding task to Transport.

Tips

- Encourage doodling: It's about insight, not art.
- Ask questions while they draw: "And what happens after that?"
- Let silences happen: This is often thinking time.
- Start rough: Only zoom in on details when necessary.
- Ask for exceptions: "What happens if this doesn't go according to plan?"
- Capture it: Always take a photo of the drawing. Even a quick sketch made on the fly will come in handy later.

What Not to Do

- Too much detail: If you draw every mouse click, you lose the big picture.
- Trying to make it too pretty: The attention shifts to the form instead of the content.
- Filling in the gaps yourself: Only draw what the other person says, otherwise you lose their perspective.

Behaviour-Oriented Probing

People love to tell you what they want: "It needs to be easier, faster, and more user-friendly." If you, as an analyst, immediately shout: "Okay, let's build it!", you miss crucial information. Because what does 'user-friendly' actually mean to this specific person?

Behavior-oriented probing means taking a step back: moving from desire to reality. You don't start with the idealized solution, but with what people are actually doing right now. How do they work? What goes wrong? What are they trying to achieve? People are often so used to their own workarounds (shortcuts or detours) that they no longer see them as problems until you ask about them.

How Do You Do It?
The key is to focus on factual behavior rather than opinions.

- "When does it go wrong now?" (Look for the cause, not the drama).
- "What are you trying to accomplish at that moment?" (Look for the intent or the goal).
- "What specifically isn't working then?" (Identify the gap between desire and reality).
- "What would you like to happen?" (Define the desired situation in terms of behavior, not solutions).

When to Use This?
Use this technique as soon as people start using abstract terms or jumping straight to solutions:

- "It needs to be easier."
- "We want a dashboard."
- "There should be a button here."

As soon as you hear a solution offered without context ("We need a dashboard for that"), a red flag should go up in your mind: Get back to the behavior!

Example: The Callback Requests

A helpdesk employee says: "I want to quickly see which callback requests are open for me."

You probe further:
- "When do you do that now?" → "Every morning when I start."
- "What exactly do you do then?" → "I open the system, filter for 'open,' and then I have to click on every single request to see if a colleague has already picked it up."

- "What goes wrong there?" → "It takes me far too much time to see who I actually need to call."
- "What do you want to happen?" → "Just a simple list of names that aren't assigned to anyone yet."

Now you no longer have a vague wish, but a concrete goal.

Tips

- Listen truly: Don't have your answer ready before they've finished speaking.
- Have them show you: Let someone share their screen and think aloud while they work.
- Verify: Summarize what you hear and see ("So, you have to click five times right now?")

What Not do Do

- Suggesting solutions immediately: The moment you start solving, you stop listening.
- Asking closed questions: "Is it difficult right now?" (A simple Yes/No doesn't provide the insight you need).
- Making assumptions: "I already understand the process." (Reality is often very different from the official theory).

The 5 Whys

Imagine someone wants a "confirmation button." It sounds logical. But why? Is it because they forget to do something? Because they feel insecure? Because their manager wants to see proof? Without the "why," you don't know if that button actually solves the real problem.

The concept of the 5 Whys is simple: you keep probing for the reason behind the reason until you reach the core. Theory says "5 times," but sometimes you get there after only 3. It's not about the count; it's about peeling back the layers.

Similar to Behavior-Oriented Probing?
Yes, they have a lot in common. In both cases, you are looking for what is really going on, rather than accepting the first answer. However, there is an important difference:

- Behavior-Oriented: What is happening concretely? (The facts).
- 5 Whys: Why is this important? (The motivation).

Together, they are a powerful duo. Use behavior to make the problem visible, and use "why" to understand the reason behind it.

How Do You Do It?
Warning: If you literally ask "Why?" five times like a toddler, you will become annoying. So, don't do that.

This technique requires patience and a curious attitude. You will often receive a superficial answer or a solution first, but that is rarely the true motive. By continuing to ask questions in a calm and friendly manner, you move closer to the core step by step.

Vary your questions:

- "What makes this important to you?"
- "What happens if we don't do this?"
- "What is the consequence of that?"
- "And why is that a problem?"

With these types of questions, you force the other person to think beyond their first impulse. Focus on the motive behind the wish, not the wish itself.

When to Use This?

- When someone offers a ready-made solution ("We want a report").
- When terms remain vague ("We want more control").

Voorbeeld: Het dashboard

A call center manager says: "I want a new dashboard."

1. You: "What do you want to see on it?" → "The progress of the tickets."
2. You: "Why is that important?" → "So that I can see if we are on schedule."
3. You: "What happens if you don't see that?" → "Then I only hear after the fact that customers were helped too late."
4. You: "And what is the consequence of that?" → "Then we lose the trust of our major customers."

Conclusion: It's not about a pretty dashboard with graphs. It's about preventing delays to maintain customer trust. A proactive notification in case of a delay might be better than a dashboard.

Tips

- Be genuinely curious: Your tone and attitude make the difference between showing interest and conducting an interrogation.
- Don't fill in blanks: The answer can be different from what you expect.
- Identify the core: "Aha, so it's actually about customer trust."
- Combine with behavior-oriented probing: Behavior makes the problem visible; "why" provides the underlying reason.

What Not to Do

- Literally asking "Why?" 5 Times: That only works with toddlers.
- Stopping too soon: Don't be satisfied with the first answer you receive.
- Sounding judgmental: “Why would you do that?” sounds like you’re judging.

Focus on Emotions

In business conversations, we often ignore emotions. We want to be professional. But frustration, irritation, pride, or even despondency are not "noise." If you want to know what people truly find important, these are vital indicators.

Emotion-oriented probing is about picking up signals. A deep sigh, an eye roll, a sarcastic joke. These are not disruptions to the conversation; they are (often unconscious) pieces of valuable information that you can dig into.

How Do You Do It?
Pay attention to more than just words. Do you notice something in the tone, the subtext, or the body language? These are all expressions of emotion. Make them discussable and link them directly to the work.

- Ask where the emotion comes from: “What makes this so annoying?”
- Ask about positive emotions too: “When does your work feel valuable?”
- Link the emotion back to the process: “What does this mean for your workday?”
- Stay professional: Keep it focused on the work and stay out of someone’s private problems.

Pay close attention to non-verbal signals and keep it work-related.

When to Use This?
Use this when you notice irritation or frustration. Behind every frustration lies a desire for how things could be better. Emotions give you insight into workload, inefficiency, and things people normally don't say out loud.

Emotion provides you with:

- Direct visibility into what is truly important to someone.
- Insight into workload, inefficiency, or poor collaboration.
- Leads for what needs improvement, which are often left unsaid.
- The opportunity to truly put the user at the center.

Example: Monday Morning

An employee mentions offhand: "Only on Monday morning I know what I need to do for my customers in that week. I'm so tired of it."

You probe further:

- "How does that affect you?" → "I lose the entire morning creating my weekly schedule. It feels like my time isn't being taken seriously."

You learned that there is an issue with workload and appreciation. This could be an essential nuance for the solution.

Note: You Are Not a Therapist
It can be tempting to take on the role of a counselor or coach as soon as someone shows emotion. That is not your job. You don't need to conduct a therapy session or uncover someone's private life. You are solely concerned with the experience within the work.

Do not wander too far down personal side paths. If someone starts talking about a divorce or illness, acknowledge it briefly and respectfully ("I'm sorry to hear you're going through that"), but then gently steer the conversation back to the work process. Don't ask about trauma; ask about how the work feels. This allows you to remain professional and guard your own boundaries.

Tips
If you take emotions seriously, you often bring the deeper need to the surface.

- State what you see: "I notice that this topic frustrates you."
- Be respectful: Give the feeling space, but maintain the link to the work.
- Ask about both sides: Don't just focus on what drains energy, but also on what provides it.
- Use silence: If someone falls silent for a moment, it's often a sign they are thinking. Don't fill that silence too quickly.
- Check your assumption: "I get the impression this makes you angry, is that correct?" Don't guess.

What Not to Do
Emotions are powerful, but mishandling signals can damage the conversation.

- Dismissing emotions: Thinking "Oh, don't be such a drama queen" is easy, but then you miss what is truly important to that person.
- Playing therapist: Again: stick to the process analysis, not the private life.
- Only noting the frustration: "Pete is angry" is not an analysis. Translate it to the process or behavior ("The process fails at step X, causing frustration").

- Only asking about negative emotions: Positive experiences (pride, joy) are just as valuable for knowing what to keep.
- Ignoring non-verbal signals: Someone who says "yes" but shakes their head or looks away likely means no.

Sketching a Future Vision

Sometimes a conversation gets stuck in frustration. People can tell you in great detail exactly what isn't working, but they find it difficult to articulate what they do want. The 'future question' is a crowbar technique that helps to break free from the daily grind and dream ahead.

By sketching the future, you invite the other person to imagine that the problem has already been solved. This forces them to distance themselves from current limitations and think in terms of effect, impact, and feeling. You don't ask: "What do you want built?" (the technology), but: "Suppose it works perfectly soon, what will you notice then?" (the effect).

For you as an analyst, this is incredibly valuable. From that perfect picture, it is much easier to reverse-engineer:

- What is needed to achieve that feeling?
- Which functions truly contribute to that?
- And what should the system definitely stop doing?

How to Do It
You ensure that people express what they have in their minds. Give them the space to fantasize, but try to make it concrete again. Use these questions as a guide:

- **The magic question**: "Suppose, in three months, this works perfectly. What can you do then that you can't do now?" (This leads to concrete wishes).
- **The impact**: "What specifically do you notice about your workday?" (This brings the value into view).

- **The visualization**: "What do you see on your screen then, without having to search?" (This makes the desired situation visible).
- **The ballast**: "What will no longer be necessary?" (This helps you discover what nonsense can be stripped from the process).

When to Use This
Use this technique if:

- People keep lingering in complaints or vague solutions ("I want a button that does that"), but fail to formulate clear requirements.
- You have the feeling that the true need is still not on the table.
- You are looking for direction for your prioritization or visual design.

Example: The expense claim

A finance employee complains loudly: "For a single expense claim, I have to go through four systems. It's so irritating and prone to error."

You ask the future question: "Suppose it works perfectly soon. What does that look like?"

The answer: "Then I open one screen, drag the receipt into it, and it's processed immediately. No more separate Excel sheets, no more double checks."

The analysis: You now have a clear future vision that provides direction. The goal is not 'building better integrations,' the goal is: one screen, drag, done. That stands for simplicity, speed, and fewer errors.

Tips
With a future question, you often discover the deeper desire behind all the complaints.

- Help people get started: People sometimes find this question a bit abstract or "vague." Provide an example yourself to break the ice.
- Ask follow-up questions: Don't settle for "It will be nice." Probe further: "What makes that important?" or "What does that yield?"
- Use it as a test: Use the future vision later to check if your design is correct. ("Does this design help to achieve that feeling?").
- Identify the value: Say out loud what you hear: "So you are mainly looking for speed and overview."
- Keep it concrete: Always bring the dream back to behavior or the workday.

What Not to Do
Future questions are powerful, but if asked incorrectly they remain dreams.

- Asking too vaguely: "Suppose everything is perfect..." is too broad without a concrete subject.
- Accepting vague answers: "Then it works immediately" says nothing about what is needed. Ask: "How do you envision that, then?"
- Only collecting dreams: Without translating them into actions and requirements, it remains just a dream. You must make the translation.
- Being satisfied too quickly: The first answer is often superficial. Dig deeper.
- Providing no support: Some people block when asked to 'imagine.' Give them concrete hooks or scenarios to work with.

Ask About Previous Solutions

In almost every project, you hear the same statements: “This just isn’t working anymore” or “We really need a system that solves this.” It sounds logical and urgent. But before you dive straight into solutions, there is one question you must always ask: “What have you already tried to address this?”

The answer is often surprisingly valuable. You might discover that a similar system was introduced before, but nobody used it. Or that there were great ideas that stalled because the organization wasn't ready for them. Sometimes something turned out to be technically unfeasible; other times it failed due to resistance or a lack of support.

This technique is about retrieving history. You dive into the past of previous solutions, not to hold people accountable for what went wrong, but to learn from those attempts.

When you know what people have already tried, you don't just discover failures; you also find:

- What they find important: Because that’s what they tried to organize.
- What they found irritating: Because those parts didn't work well or weren't used.
- The constraints: Where do the technical or organizational boundaries lie?

And perhaps most importantly: you prevent yourself from coming up with a “brilliant idea” that someone already tried a year ago and that flopped spectacularly.

How Do You Do It?
You ask about it explicitly and explore what happened together. Start simply:

- "Have you tried to tackle this problem before?"
- Follow up with deeper questions:
- The stakeholders: "Who was involved back then?"
- The reason: "Why did it stop or fail?"
- The nuance: "What worked well at the time, and what didn't?"
- The experience: "What did the users think of it?"
- The solution: "What do you think would make it workable now?"

Provide space to share failures without judgment. Many people find it slightly uncomfortable to talk about old, failed projects when there's a new chance for a solution, even though that information is incredibly valuable to you.

When to Use This?
While it's always good to ask about previous solutions, use this technique especially:

- When someone mentions a wish for something that used to exist ("We used to have…"). Why is it gone now?
- On topics where there is a lot of resistance or skepticism ("Here we go again…"). This is often due to past negative experiences.

Example: Another SharePoint?

You are analyzing an internal knowledge-sharing system. An employee says: "We really need a new platform where people can share their knowledge."

You ask: "Have you tried that before?" Employee: "Yes, we had SharePoint. But nobody used it."
You probe further: "Why not?" Employee: "You had to manually turn on notifications to get updates. Nobody knew that. And if you searched for something, you never found it."

The analysis: Now you know that the wish isn't simply "a platform" (because they already had one). The real requirements are:

- Automatic notifications for new updates.
- A powerful search function.

With this insight, you move forward. You prevent the mistake of simply putting down "just another SharePoint."

Tips
By asking this question, you show respect for what has already been done and learn from history.

- State your intention: Mention that you are not judging, but that you want to understand how to make things better.
- Look for silver linings: Ask about successes as well. What *was* good about the previous attempt?
- Look broader: Pay attention to political or organizational reasons. Sometimes the solution was good, but the timing was wrong.
- Check the perception: Did the previous failure create a negative image for these types of projects?

What Not to Do
Retrieving the past is valuable, but if handled incorrectly, it triggers resistance.

- Judging: Never say: “Well, it’s obvious why that didn't work.” People will clam up.
- Only looking at failures: You’ll miss the components that did have value and that you might be able to reuse.
- Ignoring organizational causes: Sometimes something failed due to lack of time or budget, even though the idea itself was great.
- Being arrogant: Don't be too quick to think your idea is better. Without checking, you risk making the exact same mistake.

Suppose It Failed

You are sitting at the table with a customer. There is an idea, a plan, and everyone is enthusiastic. The atmosphere is great. And then you ask that one unexpected question: “Suppose this project has failed miserably six months from now. What went wrong?”

It sounds gloomy, and perhaps even a bit like a buzzkill. But make no mistake: this is one of the most valuable questions you can ask.

This technique (also known as a pre-mortem) is about consciously triggering doubt. When people make plans, they usually look at the best-case scenario: the ‘happy flow.’ They envision how it will work and how beautiful it will look.

Because of this, risks, blind spots, or bad experiences from the past remain under the radar. By explicitly asking what could go wrong, you make that shadow side discussable. It gives you direct visibility into organizational, technical, and human risks. Moreover, it shows that you are not blindly optimistic, but a realistic partner thinking about success.

How Do You Do It?

You use questions that invite the other person to voice their concerns out loud. You do this in a friendly, open way. Not as if you hope things go wrong, but because you want to prevent them from going wrong.

Questions you can ask:

- “Suppose this system is live and yet it doesn't work. What went wrong?”
- “What could we be overlooking at this moment?”
- “Is there anything are you secretly a little worried about?”
- “What has gone wrong in similar projects in the past?”

The goal is not to scare anyone, but to understand the risks living in the minds of the people at the table.

When to Use This?
Timing is important with this technique. Use it as soon as a direction, plan, or idea begins to emerge.

- Not too early: During a brainstorm, it is paralyzing and kills creativity.
- Not too late: When the choices have already been made.

The best moment is when everyone is just starting to get enthusiastic and the plans are taking shape. Ensure a quiet setting where people feel free to speak. Don't do this quickly in passing or at the end of a rushed meeting.

Example: The Self-Service Portal

An organization is working on a new self-service portal for employees. Everyone is happy; the design looks sleek.
You ask the question: "Suppose this goes live, but it isn't used. What went wrong?"

The answers follow immediately:

- "Then it's probably too complicated."
- "Many of our people aren't tech-savvy at all."
- "Or people don't even know it exists, just like with the previous tool."

In a five-minute conversation, you discover three crucial risks:

1. If it isn't intuitive, people drop out (UX design).
2. There is a need for clear explanation or training (Education).
3. Internal communication must be in order (Communication).

With those insights, you can now adjust, even before the build begins.

Tips
If applied correctly, this technique yields a list of real points of interest that make your analysis much stronger.

- Stay positive in your wording: Explain that these are scenarios nobody wants, and that is exactly why you are discussing them now.
- Probe further: "What exactly makes that a risk?"
- Analyze the realism: Not every concern is justified. Filter out the real risks.
- Discuss the solution: Can we mitigate this? And if so, how?
- Determine the impact: Does this affect your analysis? Do you need to add an extra requirement or function?

What Not to Do
The power of this technique lies in the execution.

- Using it too early: This kills any enthusiasm immediately.
- Staying stuck in negativity: The conversation should be realistic, not somber or cynical.
- Gathering risks but doing nothing with them: This is fatal for trust. People will feel they aren't being taken seriously.
- Treating everything as equally important: Distinguish between 'small bumps' and 'showstoppers.'
- Discussing it too hastily: In a quick meeting, people often don't dare to share their real concerns. Take the time for it.

Summarizing and Verifying

You think you understand each other, but a week later it turns out you both meant something entirely different. A waste of time, and even worse: a waste of trust. This is precisely why this is one of the most powerful techniques you can use as an analyst.

Summarizing and verifying means giving back the core of what someone has told you in your own words and immediately checking if that is correct. It seems simple, but the effect is profound. People often think they are communicating clearly, while language is always vague and context-sensitive. By paraphrasing, you make implicit assumptions explicit. Furthermore, the other person feels heard, which enormously strengthens the collaboration.

Why This Works

- Focus: You force yourself to listen actively.
- Control: You directly check yourself to see if you understood the message correctly. If you cannot retell it in your own words, you haven't understood it yet.
- Prevention: You prevent misunderstandings later on.
- Connection: You let the other person feel: “Hey, they are really listening to me.”

How to Do It
Listen actively and then summarize. Use the other person’s words, but not literally. Translate it into your interpretation and make it concrete.

Say, for example:

- “If I understand correctly, you want to be able to see on a single screen who still needs to take action?”
- “So essentially, you want the system to warn you if a step remains idle for too long?”

- "You mean: the customer does receive a confirmation, but the internal team is not yet notified automatically?"

Afterward comes the most important step: the verification. Ask explicitly:

- "Is that correct?"
- "Am I missing anything?"
- "Is that what you mean?"

Help People Structure
Note: people sometimes find it difficult to explain something. Even then, this technique is handy for getting things sharper. You help them organize their thoughts. Say: "So if we look at this step-by-step: first X happens, then Y, and then it gets stuck at Z. Is that right?"

Observe the Silence
Sometimes it remains quiet for a moment after your summary. Let that happen. Silence is often the moment when someone considers whether your summary truly covers the essentials. Do you see hesitation in their eyes? Seize that opportunity:

- "I see you're hesitating?"
- "What doesn't feel quite right yet?"

When to Apply This
Use summarizing and verifying at these moments:

- At the end of an explanation.
- When you notice someone is drifting off or remains vague.
- When you start to doubt whether you understand it correctly yourself.
- When you feel you are *not* quite understanding each other, but no one is saying it out loud.

It may seem simple, but it is a crucial skill. It provides clarity, prevents misunderstandings, and often yields new insights. Better done too often than too little.

Example: Active warning or not

A customer tells a long story about a new requirement. You summarize: "So, if I understand correctly, you mainly want the system to actively warn you as soon as the manager's approval takes too long?"

Scenario A: The customer nods enthusiastically. "Yes, exactly that!" (Check, you're on the right track).

Scenario B: The customer says: "No, that's not quite what I mean. I want to see it, but I don't need a warning." (Check, you have prevented a misunderstanding).

By actively verifying, you prevent unnecessary work.

Tips
Use this technique as much as possible. Doing this consistently strengthens both the content and the relationship.

- Paraphrase: Use your own words, not the other person's. This is the only way to check if you understand it.
- Always ask explicitly if it's correct: Don't just assume.
- Use silences: They give the other person space to think.
- Help them search: Summarize step-by-step if someone is still searching for the right words themselves.
- Frequency: Do it more often rather than less to prevent mis-understandings. Even if it all sounds straightforward.

What Not to Do
Summarizing seems easy, but if done incorrectly, it yields little value.

- Parroting: Literally repeating what the other person said. This adds nothing and can be irritating.
- Skipping it: Thinking "I understand it well enough." That is precisely when the biggest mistakes occur.
- Failing to check: Without the question "Is that correct?", your summary is worthless.
- Summarizing too superficially: Ensure you capture the depth, otherwise the nuance is lost.
- Treating it as a formality: It is a conversation technique to gain understanding, not a checklist to tick off.

Techniques to Structure Your Work



Techniques to Structure Your Work

Holding conversations is the foundation of your work. With good questions, probing, and summarizing, you can go a long way. But you aren't there yet with just a good conversation.

Gathering information is one thing; doing something meaningful with it is another. A conversation alone doesn't produce a tight priority list. It doesn't create a shared vision of who the users are. And it doesn't make the tough calls between what is "nice to have" and what is "absolutely necessary."

For that, you need other techniques. Think of this as the contents of your toolbox. I'm not talking about heavy, theoretical models that you fill in only to let them gather dust in a drawer. I'm talking about practical ways to bring structure to the chaos you've gathered. The goal isn't to write thick reports, but to create clarity quickly.

Think of situations such as:

- Visualizing: How do you map out a complicated process so that everyone immediately sees where it gets stuck?
- Prioritizing: How do you get priority when your customer claims everything is Priority 1?
- Who matters most: How do you map out who truly matters (even if they don't see themselves as important, or perhaps think they are more important than they are)?
- Slicing building blocks: How do you break down large wishes into small blocks that deliver immediate value?

In this section, we discuss the techniques that help you achieve this. Don't see them as mandatory exercises, but as tools you can use flexibly. You just need to know which tool to grab from the box to get the job done.

The Flowchart

In the chapter on conversation techniques, I gave a few examples, but flowcharts will almost certainly be your favorite. You can use this technique anytime, regardless of the problem or where you are in your analysis. Whether you are still bouncing ideas around or doing a deep dive, this diagram always provides an overview.

A flowchart is a visual overview of a process. It shows the steps you take, the choices you make along the way, and where you eventually end up. It forces you to think logically: after step A, an arrow must lead to step B. If the arrow goes nowhere, your process is incorrect.

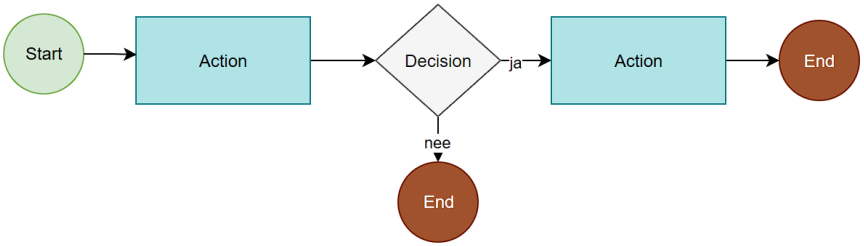
Why Does This Work for Almost Everything?
People often tell stories in a chaotic way. They jump from one thing to another: "Yeah, when the application comes in, we look at it, unless it's incomplete, oh yeah, and Pete also has to approve it, but that's not necessary if the amount is under € 500."

When you map this out in a flowchart, you immediately see the gaps:

- What happens if Pete is sick?
- Where does the incomplete application go? Back to the customer? Into the trash?

How Do You Do It?

You can make flowcharts as complicated as you want (just Google them...), but in practice, for 95% of your work, these four symbols are enough:





The Oval (Start/End)**:** This is where the process begins and ends. Every diagram has at least two of these.



The Retangle (The Action): Something that is being done. Examples include: "Fill out form," "Calculate total," or "Send email."



The Diamond (The decision): The intersection. This is where a question is asked that can be (mostly) answered with a simple Yes or No. For example: "Is the stock empty?"

The Arrow (The Direction): This indicates the flow. Where are we going next?

A good habit when creating flowcharts is to keep them simple. It is tempting to draw everything in a single diagram, but you will soon notice that full processes with all their logic and exceptions becomes enormous. It also becomes difficult to "cram in" everything you discover later.

The solution to this is **drilling down**. Every action and decision can be mapped out in a separate flowchart, where you detail the deeper logic within that specific block. For instance, you could draw a new flowchart for an action like "Approval" that describes the logic behind the approval

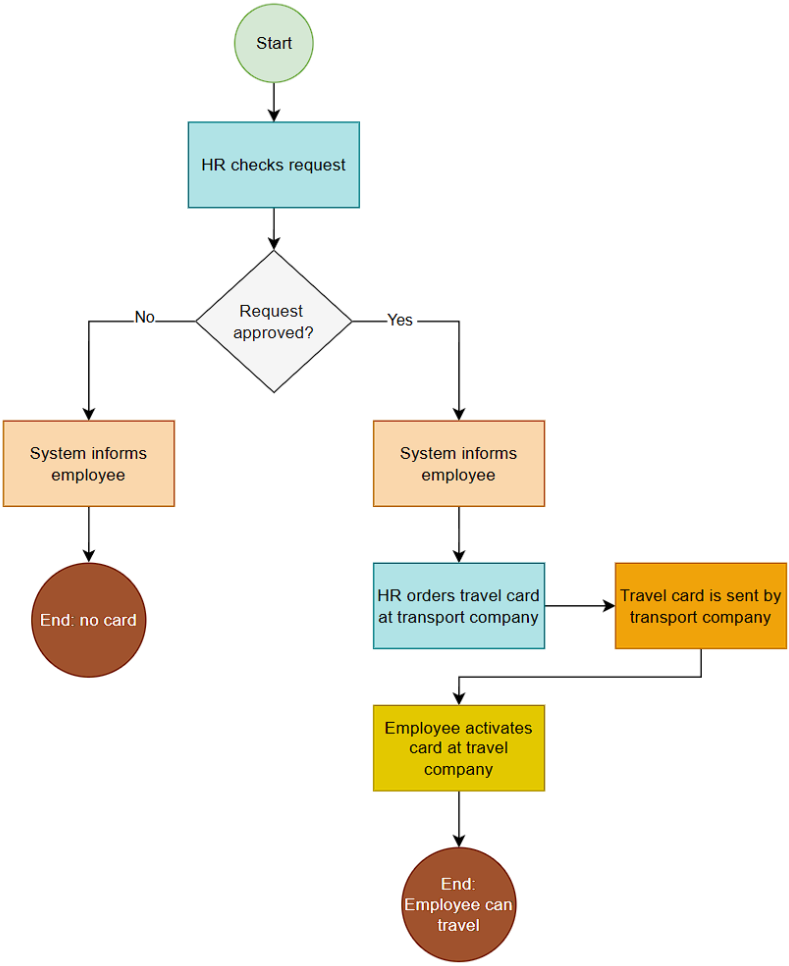
process while keeping it simple in your master flowchart. You can find an illustration of this in the example scenario of this chapter.

Example: Approving the Public Transport Card

HR needs to approve public transport card requests for employees. You map it out:

- Start: Request is received.
- Action: Check request.
- Decision (Diamond): Is the request approved?
    - No → Inform employee
    - Yes → Inform employee, order card, send card, activation by employee
- End: No card or card received.

On paper, this sounds clear and simple, but mapping it out reveals a flaw that would likely have been overlooked.

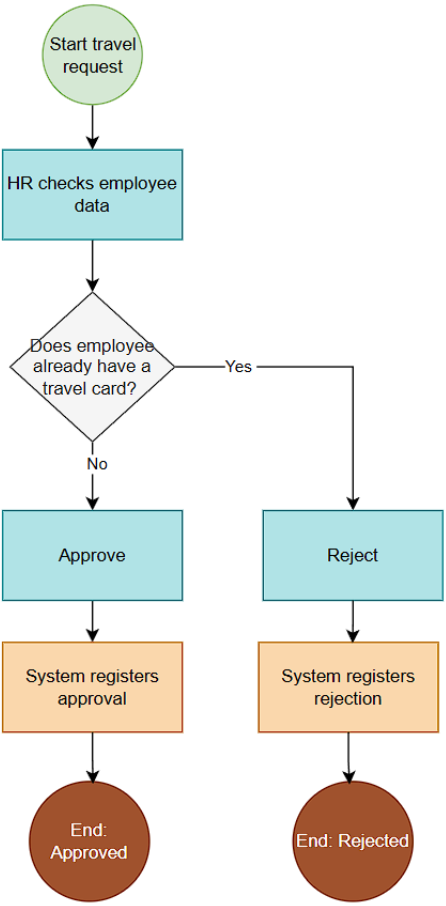


Did you spot the flaw in this diagram? We aren't recording the outcome anywhere. This means the employee can keep submitting requests, and HR will have to keep checking them over and over.

Your first instinct might be to simply add more blocks to the diagram, but don't do that unless you're certain the drawing will remain small.

You will soon find that with all the ifs and buts, your diagram will quickly become cluttered and oversized.
The solution is drilling down. You create a separate, detailed breakdown for the "Check request" block. In that sub-process, you specify that the status ("Approved" or "Rejected") is stored.

Problem solved, without a messy main drawing.



When zooming in, always ask yourself: Have I covered everything now? The example above was intentionally kept simple, but reality is often full of ifs and buts. For instance: Does this apply to all salary grades? Are there different types of subscriptions? Can an employee switch mid-term? What happens during job changes? Shouldn't we register the exact date the card is put into use?

It is precisely in your deeper breakdowns that you must capture these details. This keeps your master drawing readable while ensuring your analysis is complete.

Tips

- Does it flow smoothly? A good diagram is read from top to bottom or from left to right. Avoid zigzagging all over the paper.
- Yes/No is sacred: Ensure that a Decision (Diamond) always results in clear, outgoing lines. Usually, "Yes" and "No" are sufficient. If you have more options, use multiple diamonds in succession.
- No dead ends: Every line must lead somewhere. A process never simply stops in thin air (unless it is the "End").
- Maintain the overview: Work out the main process first, and only then focus on the logic behind each individual block.
- Use colors: In a flowchart, multiple people or systems often perform actions; by giving them their own color, it quickly becomes clear who does what.

What Not to Do

- Processing everything in one drawing: Do not try to include every single step in detail in one diagram.
- Crossing lines: Try to avoid arrows crossing each other, as it becomes unreadable. Move your blocks until the lines run freely.
- Too much text: Do not put entire paragraphs in a block. Use verbs: "Approve request," "Send email," "Update status."

MoSCoW: When Everything Seems Important

In almost every project, you encounter the same problem: everything is important. One stakeholder wants to input data, another wants automatic processing, and a third wants a slick report. Before you know it, you have a wish list where every line is marked "Priority 1." If everything is a priority, nothing is a priority.

MoSCoW is a simple but powerful technique for making tough choices together. You divide all wishes into four categories:

- **M - Must have**: Without this, the solution simply won't work. If this is missing, the project fails. (Think of legal requirements or a process that stops without this step).
- **S - Should have**: Very important. If this is missing, it's painful or difficult, but a (temporary) workaround is possible.
- **C - Could have**: Also known as 'Nice to have.' We only do this if there is time and budget left over.
- **W - Won't have** (for now): We consciously decide not to do this in this phase. Perhaps later, but not now.

It may seem like a simple model, but the effect is profound: you take the emotion out of the discussion by collectively determining what is truly necessary and what is merely 'desirable.'

How to Approach This
MoSCoW only works if you facilitate the conversation properly. It's not a box-ticking exercise in Excel; it's a negotiation. Here is how you do it:

1. **Explain the rules of the game**: Be strict. A 'Must' is truly a Must. If we mark everything as a Must, we will likely deliver nothing.
2. **Let everyone think for themselves first**: Prevent the loudest voice from dominating the direction.

3. **Discuss the differences**: Why does one person see something as a Must while another sees it as a Could? This is often where the most valuable information lies.
4. **Check the balance**: If 80% of the wishes are 'Musts,' you haven't actually set priorities. Force the group to choose.
5. **Make it visible**: Use a whiteboard or a digital wall with sticky notes. When you physically move a card from 'Must' to 'Should,' everyone feels the weight of that choice.

When to Use This

- When there are too many wishes for the available time or budget.
- At the start of a project to manage expectations.
- When tensions rise between stakeholders ("My part is more important!").
- During the refinement of the backlog, when it looks like everything needs to be done at once.

Example: The Customer Portal

A project team is working on a new customer portal to provide insights into energy consumption. The stakeholders provide a laundry list of wishes: logging in, a chat feature, multi-language support, viewing energy contracts, energy analysis, and a dashboard.

In the first round, everyone shouts: "Must!" so you facilitate the conversation w ith the critical question: "Can the product go live without this component?"

- "If there is no overview of current energy consumption, can the customer still use the portal?"
    - Answer: "No, because that's what most customers come for."

- Conclusion: → Must have

- "Is viewing the energy contract a core goal of the portal?"
  - Answer: "The goal is providing insight into energy usage. However, as an energy supplier, we do want to make the current contract visible in the portal."
  - Conclusion: The goal is energy insight, not contract management. → Should have
- "Are all languages equally important?"
  - Answer: "English and Dutch are necessary. German would be nice for people in the border regions."
  - Conclusion: NL/EN → Must, German → Could or Won't

You go through all the wishes and requirements in this manner. Within an hour, you have shifted the discussion from "I want this" to "What is necessary?"

Tips

- The litmus test: If in doubt, always ask the question: "If we leave this out, will it cease to function entirely?" If the answer is "Yes, but...", then it is not a Must.
- Be strict: Dare to engage in the discussion. Not everything can be equally important.
- Don't forget the 'Won't': This category is crucial for expectation management. By explicitly placing something in the 'Won't' bucket, you are saying: "We've heard you, but we aren't doing it right now." That is far better than ambiguity.
- Repeat the process: Priorities shift. What is a Could today might become a Must next month due to new legislation or changing market conditions.

What Not do Do

- Putting everything in Must: If you do this, you might as well not use the model at all. It defeats the entire purpose of prioritization.
- Avoiding a collective discussion: If you fill it in alone, no one will feel ownership of the choices. You need the stakeholders to agree so they support the final decision.
- Confusing it with scheduling: MoSCoW is about importance (priority), not about sequence (planning/scheduling). Sometimes you complete a Should have before a Must have because it is a "quick win" or removes a dependency for later.

Personas: Because “The User” Doesn’t Exist

Systems are built for people. But "the user" does not exist. An administrative assistant who enters data all day works completely differently than a manager who runs a report once a month. And a digital novice needs different support than a power user who knows every keyboard shortcut.

Yet, in projects, you often hear: “The user wants this.” This is dangerously vague. Which user are we talking about?

Personas help break through this vagueness. You create a number of concrete, fictional (but realistic) profiles that represent your most important target groups.

Note: These are not marketing caricatures (“Linda, 34, loves cappuccinos, yoga, and lifestyle magazines”). For information analysis, that is useless info. You need a tool to clarify behavior, goals, and frustrations.

How to Approach This
A good persona is based on observations from practice, not on what you make up at your desk. You don’t need to write a novel; half a page per profile is enough.

1. **Gather input:** Talk to various users. Note their goals and daily tasks, but especially their frustrations and digital literacy.
2. **Clustering:** Look for patterns. Which groups are similar in behavior or needs? (Sometimes a manager behaves like a new employee because both hardly know the system).
3. **Create profiles:** Give each group a name and a short profile. Describe their context, their goal, and their biggest **pain point**.
4. **Use imagery:** Attach a photo or an icon. It might sound childish, but it makes a persona human during discussions.
5. **Keep it relevant:** Only write down what is relevant to the design or the prioritization.

When to Use This

- At the start: To get a sharp image of exactly who you are doing this for.
- During work sessions: As soon as the discussion gets bogged down in "Yeah, but the user wants...", you cut it off with: "Which user do you mean?"
- As a counterweight to management: If decisions are only being made in the boardroom, personas help bring the front line back into focus.
- During prioritization: A feature might be a Must for Persona A, but merely noise for Persona B. This helps in making tough choices.

Example: Renewing the Inventory System

An organization is working on renewing its existing inventory system. Everyone keeps talking about "the employee." You notice this is too vague and create three distinct personas:

- Sandra: A back-office employee. Recently hired. Doesn't know the program well. Easily gets lost in the menu.
- Pete: A back-office employee. Has worked here for years. Uses the system daily. Knows all the shortcuts by heart and is incredibly annoyed by every extra mouse click or duplicate entry.
- Bob: A manager. Experienced with the system but uses it infrequently. He needs to see the stock status at a glance. When he logs in, he requires an immediate overview of relevance.

With these personas, discussions become immediately concrete. If your client says, "The navigation needs to be simpler," you can ask: "Do you mean for Sandra, who doesn't know her way around? Or for Pete, who just wants to work as fast as possible?" For Sandra, you might design onboarding tooltips or a guided interface. For Pete, you focus on keyboard shortcuts and a high-density layout.

Tips

When used correctly, personas bring users to life.

- Keep it brief: A half-page with bullet points is perfect.
- Make it visible: Literally hang them on the wall or put them on the first slide of your presentation.
- Validate them: Let real users read the personas. Do they recognize themselves or their colleagues?
- Use the names: Refer to ‘Sandra’ and ‘Pete’ in meetings instead of ‘the user.’
- Choose the right focus: Focus on behavior and goals, not on age or job titles.

What Not do Do
Personas are only useful if you use them correctly. Don't fall in these traps:

- Creating marketing profiles: Age, hobbies, and favorite TV shows are rarely relevant to your solution.
- Too many personas: Usually, 3 to 5 clear profiles are enough to cover 90% of your users. For very simple projects, you can even skip them.
- Confusing them with job titles: A persona is not the same as a job title. 'The manager' might be tech-challenged (behavior), while 'the intern' understands everything in two seconds. Cluster based on that behavior, not the title in their e-mail signature.

INVEST: The Quality Check for Building Blocks

We have discussed building blocks earlier in this book. It is the process of breaking down pieces of work into manageable parts. However, not every building block you come up with is immediately usable. Sometimes it is too large, too vague, or depends too heavily on other components.

You want to avoid "throwing a vague plan over the fence" to the execution team. To ensure that a building block is truly workable, many teams use the INVEST model.

INVEST is a handy mnemonic that stands for:

- **I** - Independent
- **N** - Negotiable
- **V** - Valuable
- **E** - Estimable
- **S** - Small
- **T** - Testable

Independent
A building block should ideally stand on its own as much as possible. If everything is intertwined (“I can’t start until Pete finishes step 3”), the planning gets stuck.

- The goal: Slice blocks so they deliver value independently or can be built separately.
- The reality: Sometimes dependencies are unavoidable (e.g., you usually can’t work in a web app without logging in first). That’s fine, as long as you explicitly identify that dependency.

Negotiable
A building block is a starting point for a dialogue, not a contract set in stone. You describe the "what" and the "why," but you leave room for the "how."

- The goal: Give the team the freedom to choose the smartest technical solution.
- The reality: Sometimes things are fixed due to legislation or previous agreements. Be honest about that: “This is a hard requirement, not a point of discussion.”

Valuable
Every block must deliver something for the user or the organization. Avoid technology for the sake of technology.

- The goal: Always make it clear who will benefit and why.
- The reality: Sometimes you have to build something technical (like security). Translate that into value: “We are building this to prevent data leaks,” instead of “We are installing a firewall.”

Estimable
The team must be able to say: “This will take us about three days.” If they say, “We have no clue whatsoever,” your block is still too vague.

- The goal: Be concrete enough for a realistic estimate.
- The reality: Sometimes something is brand new and complex. That’s allowed, but agree to perform a small "research spike" first to gain more clarity.

Small (Enough)
The smaller, the better. Small blocks provide speed, overview, and faster results.

- The goal: Blocks that can be completed in a few days (or even hours).
- The reality: Some wishes are simply large. Try to slice them anyway. Can the first version be simpler?

Testable
When is it ‘done’? You must be able to objectively determine if it works.

- The goal: Concrete Acceptance Criteria. “If I do X, then Y must happen.”
- The reality: Not everything is black and white. In those cases, make agreements on what is ‘good enough’ so you don’t end up in endless discussions.

A good building block doesn't have to score a 10 on every point, but the more checkmarks you can give it, the greater the chance that your plan is truly executable and valuable.

How to Approach This
INVEST is not a form you have to fill out. See it as a lens through which you view your own work.

1. **Write first, check later**: Just write out your building block first. Keep it short and clear.
2. **Test yourself**: Read your text back and ask yourself the question for each letter. Is this block small enough? Is it testable?
3. **Note doubts**: If the answer is 'no' or 'maybe' for any point, write down why.
4. **Improve immediately**: Is it too big? Slice it. Is it vague? Make it concrete.
5. **Dare to scrap**: If a block adds no value at all, get rid of it.

The result: a building block that not only you understand, but that the entire team can actually work with.

When to Use This

- During your own analysis: Use INVEST to test your concepts before you share them.
- During team meetings: If the team is in doubt, bring up INVEST. "Guys, is this Small enough? Or is it not Testable?"

Example: Managing the Profile

A team discusses a building block: "As a user of the internet portal, I want to be able to manage my profile so I can update my own data."

The INVEST check:

- Independent – No, it is linked to logging in.
- Negotiable – Yes, there is room for discussion.
- Valuable – Yes, useful for the user.
- Estimable – No! Far too vague. What does 'manage' actually mean?
- Small – No! This could include almost anything, so it is likely quite large.
- Testable – No! When is it considered 'done'?

It is decided that this building block contains too many ambiguities, so it is split into three small, clear blocks:

1. Change password.
2. Update address details.
3. Upload profile picture.

Now they are Small, Estimable, and Testable.

Tips

- Use it as a starting point: It should open a conversation, not close a discussion.
- Be strict about 'Small': Oversized blocks are the number one cause of delays.
- Make 'Negotiable' concrete enough: There must still be a clear goal.
- Together is better: INVEST works best when the entire team contributes and thinks along.

What Not to Do

- Just ticking boxes: It's not a checklist. It's about the insight you gain during the process.
- Being too strict: 100% perfection doesn't exist. Good enough is good enough.

Artificial Intelligence: Your Assistant

I deliberately chose not to mention Artificial Intelligence (AI) until now because it is crucial that you first understand how information analysis works and learn a solid approach, rather than outsourcing everything to AI immediately.

Think of AI tools (such as ChatGPT, Claude, Copilot, and Gemini) not as your replacement, but as your assistant. It is fast, possesses (theoretical) knowledge on almost any subject, and never gets tired. However, it also lacks a conscience, does not understand your specific company culture, and occasionally invents things that simply aren't true. It is a fantastic resource, as long as you remain the boss.

How AI Can Help You

Starting an information analysis project can be daunting. Even with the approach described in this book, it can be difficult to decide where to

begin, which method is best, or if you have been thorough enough. Use AI for:

- **Brainstorming and preparation:** Do you have an interview tomorrow with a client about a new logistics process? Ask your AI assistant for help:

  "I am an analyst and I need to conduct an interview with a logistics manager about improving inventory. Provide 10 critical questions I can ask to uncover the main bottlenecks." You don't have to use all ten, but there will surely be three in there that you hadn't thought of yourself.

- **Summarizing and structuring**: Have you spent an hour taking notes while everyone was talking at once?

  "Here are my raw notes. Extract the 5 most important action points and summarize the core of the problem in one paragraph." Because you were present during the conversation, you know exactly where the AI hits the mark or where it misses the mark.

- **Drafting concepts**: Do you need to work out twenty building blocks? Let AI provide the first draft.

  "Write a User Story for a customer who wants to change their address, including acceptance criteria and error messages." The result is rarely 100% perfect, but within ten seconds, you have a foundation that is 80% there. You handle the fine-tuning.

- **The devil's advocate (Pre-mortem)**: Use AI to find your own blind spots.

  "I have designed this process for requesting leave [paste process]. What could go wrong? Invent 5 scenarios where this process gets stuck."

- **(Technical) translation**: Do you struggle to understand the terminology used by a software developer or your client?
  "Explain what a '...' is as if I am 12 years old, and use an analogy involving a restaurant."

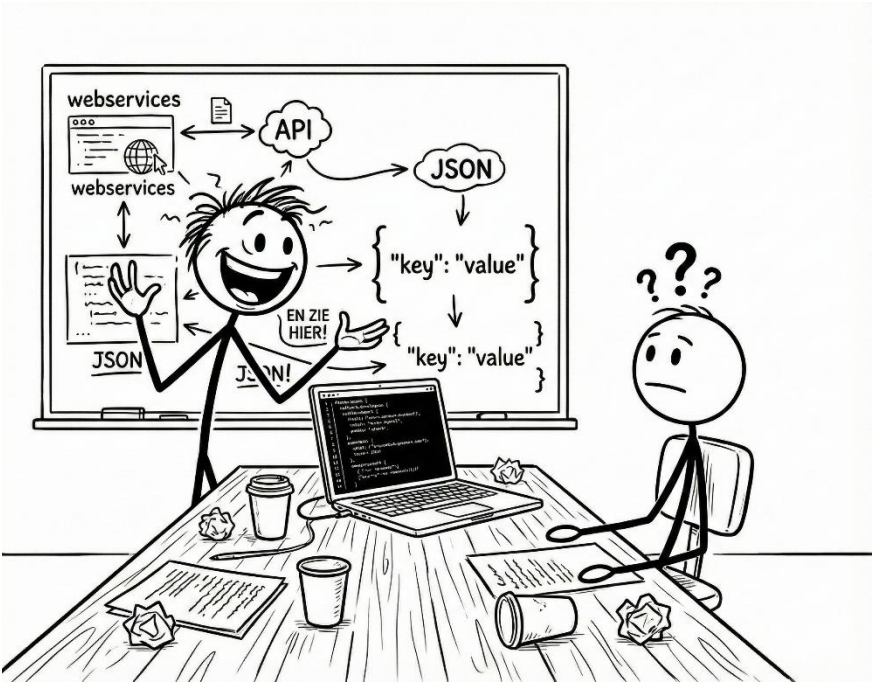
Why You Must Remain the Boss

If AI is so smart, why do we still need you? Because AI lacks three crucial things that you possess:

1. Context and feeling: AI doesn't know that Henk from administration walked out of a meeting angry last week, that there is a political battle raging between Sales and Marketing, or what has been tried unsuccessfully in the past. AI does not understand nuance or atmosphere. You do. You know when a "yes" actually means a "no."
2. Truth: AI models are trained to provide a plausible answer, not necessarily a true one. This is called hallucinating. If the AI doesn't know the answer, it might confidently invent a fact, a legal article, or a technical solution that doesn't exist. Always check the output. You are responsible for that, not the chatbot.
3. Privacy and security: Be careful with what you input. Never (and I mean never!) paste personal data, passwords, or trade secrets into a public AI tool. Anonymize everything before you ask.

In the past, the analyst was the one who typed everything out. With AI, your role is changing. You are becoming less of a writer and more of a director. You set the direction, you ask the critical questions, you check the quality, and you ensure that the message lands well with the people involved.

Basic Technical Knowledge



Basic Technical Knowledge

Not every project is technical. Not every analysis involves code or data flows. But if you work as an analyst within an IT environment, alongside developers and architects, then this chapter is essential.

Imagine you and your team are discussing your analysis. You enthusiastically explain a new feature that needs to retrieve data from another system. A developer frowns: "Do we know if that system has a REST API?"

You look at him blankly: "A rest… what?"

Another developer shifts uncomfortably in his chair: "API. Well, how do I put this... It's, uh… a sort of linking thingy."

You can only admit: "Yeah… no… I don't actually know."

In IT teams, you don't work *for* the developers, but *with* them. You sit in the same meetings and search for solutions together. Of course, they are the technical specialists, but it doesn't help if you tune out as soon as terms are dropped like:

- "We're missing the relationship between those two tables."
- "This comes in via a POST request."
- "Are we logging this in acceptance?"

If you can only stare blankly at moments like these, sharing ideas becomes difficult. You ask less pointed questions, overlook risks, and won't understand why something is technically unfeasible one time and a piece of cake the next.

In this chapter, I explain the terms you will hear every day. Not to teach you how to program, but to teach you how to understand.

API's: How Systems Talk to Each Other

Systems talk to each other constantly. A webshop talks to the inventory system, the payment service, and the carrier. But what language do they speak?

The Language: JSON and XML
Systems don't talk in full sentences, but in structured messages.
JSON (JavaScript Object Notation) is the norm these days. It is lightweight and readable. A message looks like this:

JSON

```
{
  "customer_number": 123,
  "name": "John Doe",
  "active": true,
  "orders": ["AB123", "CD456"]
}
```

The main thing to remember: it works with "**key**": "**value**".

Older systems (often in government or banks) still use XML (Extended Markup Language). Although this isn't used much in modern systems anymore, it's handy to recognize the difference.

XML

```
<customer>
  <customer_number>123</customer_number>
  <name>John Doe</name>
  <active>true</active>
</customer>
```

The Counter: APIs
Most systems exchange those JSON messages via an API (Application Programming Interface). Think of an API as a digital messenger. You give the messenger a note (a request) and they come back with a reply (a response).

You use four methods for this:

- GET: Request data (Can I have the customer details?)
- POST: Create something new (Create this new customer)
- PUT: Change something (Adjust the address of this customer)
- DELETE: Remove something (Delete this customer)

Status Codes: Did It Go Well?
With every answer, the API sends back a code. As an analyst, it's handy to recognize the most important ones so you understand what is going wrong:

- 200 OK: Everything succeeded.
- 400 Bad Request: You sent wrong data (e.g., a date in the past).
- 401/403 Forbidden: You are not allowed in (no permissions).
- 404 Not Found: What you are looking for doesn't exist.
- 500 Server Error: The system is broken (not your fault).

The Old Way: SOAP
Besides the modern API described above, you will sometimes encounter SOAP at large organizations. This is an older, formal standard that is much stricter. It works with the older XML and strict rules. I won't go into too much detail here as its usage decreases rapidly. For you as an analyst, SOAP often means: programmers have more coordination to do and have less flexibility than with REST.

Why Do You Need to Know This?
In your analysis, you determine which data is needed. If you say "we need the date of birth," that date must be included in such a JSON message. Is it missing there? Then your entire function won't work.

Databases: The Memory of the System

Without a database, a system remembers nothing. No customers, no orders, no history. At their core, most databases look a lot like Excel.

- Tables: Every subject has its own tab (e.g., 'Customers', 'Products').
- Columns: The properties (Name, Address, Price). If you don't have a column, you have nowhere to store that information.
- Rows: The actual items (Customer John Doe is one row).
- Relationships: The power lies in the link. In the 'Orders' table, you won't find the customer's full name, but only the Customer Number (e.g., 123). This refers back to the 'Customers' table.

SQL: The Query Language
Developers usually use SQL (Structured Query Language) to talk to the database.

```
SELECT name FROM customers WHERE city = “Utrecht"
```

Why Do You Need to Know This?
So you know what’s possible. If you ask: "I want an overview of all customers who bought red shoes in 2023," you need to know if that relationship can actually be made. If the color isn't stored in its own column in a table, your wish is technically difficult to fulfill without changing the database.

Environments (DTAP)

You hear it often when talking to developers: "Is it on ACC yet?" or "It works fine in PROD."

Developers don't work on just one computer, but in various environments (often called 'the DTAP street').

- D - Development (DEV): The workshop. This is where things are built. It's messy, things are half-finished, and the data is fake.
- T - Test (TEST): This is where the technical side is checked to see if it works.
- A - Acceptance (ACC): The dress rehearsal. This is where users (and you!) check if the solution is correct. This environment looks the most like reality.
- P - Production (PROD): The real system. This is where real customers work with real data. Nothing is allowed to go wrong here.

Why Do You Need to Know This?
Suppose someone says: "I'm missing a button." Your first question should be: "Which environment are you looking at?" Maybe the button isn't visible in Production yet, but it is in Acceptance. Or maybe the test-data in the Test environment is so messy that it cannot be tested properly or seems like something is broken. Know what you are looking at.

Logging: The Black Box

If a system behaves strangely, a developer will say: "I'll check the logs." Logs are like the black box in an airplane. Everything the system does (and every error that occurs) is written to text files.

Why Do You Need to Know This?
You don't have to read logs yourself, but you must know they exist.

- Is there an incident? Ask: "What do the logs say?" (This is often much better than guessing).
- Privacy and security: As an analyst, you must be sharp. Are tax ID's or passwords being written to the logs? That is absolutely not allowed (GDPR). Furthermore, log files are often shared among developers for error analysis, which happens outside of the standard security systems. If personal data is included, there is a greater risk of it falling into the wrong hands. Be critical of the content of logs.

Frontend, Backend and Microservices

Software consists of layers.

1. Front-end: What the user sees (the app, the website)
2. Back-end (of Back-end for Frontend): The brains. Calculations are made and data is retrieved here.
3. Microservices: Small, independent pieces of software that work separately but need each other to form one system.

In the past, software was one big block (a **monolith**). After that, we had a separate frontend and backend. Nowadays, we usually cut software into small, independent pieces: the microservices.

Why Do You Need to Know This?
For you as an analyst, working with microservices has advantages and disadvantages.

- The advantage: If the 'Payment service' in a webshop is broken, you can still browse the products. You know exactly what is and isn't working.
- The disadvantage: If you have a simple wish ("Show on the product page if the customer has bought it before"), the frontend suddenly has to talk to the Product service, the Order service, and the Customer service. That sometimes makes a 'simple question' technically complex.

And Then There is Also ...

To make your vocabulary complete, here are four more terms you are going to hear:

- Authentication & Authorization:
    - Authentication: Who are you? (Logging in with a password or face ID).
    - Authorization: What are you allowed to do? (Roles and permissions). If someone gets a "403 error," you now know: that's a permissions issue.
- Caching: The temporary storage of data to make the system faster. Are you seeing an old value even though you just changed something? Chances are you are looking at the cache.
- Events: Systems reacting to a signal. "If there is an order (event), then send an email." The email isn't sent directly by the 'order button,' but it reacts to the event. So, if you don't receive that email, the problem might be a broken event.
- Version Control: Team A is using version 1.2, while Team B is already on version 1.8. There may be differences between them, causing features and data to be different than expected or simply missing. As an analyst, you check: "Are we talking about the same version?"

You aren't a developer, so you really don't need to know everything in detail, but you are the **bridge between the wish and the execution**. With this basic knowledge, you can ask critical follow-up questions, understand why things sometimes take longer, and be an **equal partner** in discussions for your team.

Some Last Tips



Some Last Tips

You're almost there. You have now learned the basics, the process, and the techniques. Theoretically, nothing stands in your way of delivering a top-tier analysis. But as we all know, theory and practice are, unfortunately, not always the same.

That is why I want to conclude this book with three topics that can cause a hassle in almost every project if you don't pay attention to them.

The Forgotten Requirements (It works, but...)

As an analyst, it's easy to focus solely on the steps of the process: "First we fill out the form, then we approve it." But there could be more. You might need to establish the conditions under which this process must run.

A process can look perfect on paper but fail completely in practice because it cannot withstand reality. Therefore, ask yourself these quality-related questions:

- Time and patience: How long is something allowed to take? Can a customer wait a week for an answer, or must it be within an hour? (A process that works but takes too long is still a bad process).
- Volume and peaks: Does your solution work during a peak moment? A method where you manually look something up works fine for three customers a day, but causes chaos with thirty customers per hour.
- Access and confidentiality: Who is allowed to see what? Is the intern allowed to see everything? Does an employee have the same access as the manager?
- Error-proneness: What if someone makes a mistake? Is your solution designed so that a (accidental) mistake causes irreparable damage, or is there a safety net?

If you don't discuss these constraints, people often choose the path of least

resistance. In practice, that is often the route that leads to delays, leaks, or chaos later on.

The Art of Saying 'No'

As an analyst, you are the guardian of the boundaries. Clients often want the moon and the stars: "Everything must be possible, it must be finished tomorrow, and it shouldn't cost anything." It is your thankless task to step on the brakes occasionally. How do you say 'no' without ruining the relationship?

- Don't say 'No', say 'Yes, provided...': If someone wants a massive new feature, say: "Yes, we can build that, provided we drop something else, because our time is fully booked." This puts the choice back on the client.
- Refer to the goal: "I understand you want this, but we agreed to solve the planning problem first. This distracts from that."
- Be the expert: Sometimes a client asks for something that is technically inconvenient or unsafe. Be honest: "I advise against doing this because it makes the system slow/unsafe. My proposal is..."

Remember: an analyst who says 'yes' to everything eventually delivers a project that never finishes or is riddled with flaws. Saying 'no' is also a form of delivering quality.

Where do You Keep the Truth?

You have made beautiful analyses, drawn processes, and worked out requirements. But where do you store them? All too often, I see projects where important info stays in people's heads, or is scattered across 20 emails and three versions of a Word document called Final_v3_ReallyFinal.docx.

That is asking for trouble. Because a developer looks at version 2, the tester at version 3, and you have version 4 in your head.

Ensure there is a Single Source of Truth. It doesn't matter where this is. It can be in Word, Excel, a tool like Confluence, Jira, or SharePoint, as long as it meets two rules:

- It is centralized: Everyone looks at the same, live version. Never email documents around as attachments.
- If it isn’t there, it doesn't exist: Made an agreement at the coffee machine? Fine, but it only counts once it’s in your central truth.

Force yourself and your team to declare that source sacred. It will save you endless discussions later about "what did we agree on again?".

Finally: It's Your Turn Now

You've finished the book. You now know that information analysis isn't about writing thick reports, but about asking the right questions. You have filled your toolbox with a phased approach, conversation techniques, and methods to structure your work. You know how to make AI work for you and how to understand a developer talking about APIs. Now the real work begins. Because you don't learn information analysis from a book, but by doing it.

Start Small
Your head might be spinning a bit. "Do I have to create personas, draw a customer journey, and ask 'why' five times for every project?"

No. Absolutely not.

Think of this book as a toolbox where you can take out what you need at that moment. You grab the tool you need for the task at hand.

- Is the goal vague? Draw a process.
- Are there too many wishes? Use MoSCoW.
- Is the atmosphere stiff? Ask about the past.
- Lack of inspiration? Ask AI.

You don't have to use everything, and certainly not all at once.

Dare to Be the 'Dumbest' Person in the Room
If I can give you one last piece of advice, it's this: don't be afraid to ask the 'dumb' questions.

In meetings, people often keep their mouths shut because they are afraid they should know something. They nod 'yes' while thinking 'no.' You are the one who breaks that silence. You are the person who dares to say: "Sorry, I don't quite get it yet. Can you draw that for me?"

It feels uncomfortable at first. But you will see: as soon as you ask that question, three other people will let out a sigh of relief.

You Don’t Simply Take Notes

Remember how we started this book. You aren't there to write down what the client says, but to discover what the client needs. That difference is what makes you indispensable.

Whether you have 'Information Analyst' on your business card, or you are a project manager, developer, or office manager: the moment you stop making assumptions and start asking questions, you are analyzing.

So: close the book, grab a marker, and start asking.

Good luck!

Sources & Reading Tips

The techniques and methods I use in this book were not all invented by me. In many cases, they are existing techniques to which I have given my own, simpler twist.

For those who want to go deeper or read the official documentation, I have gathered the most important sources and my personal reading tips below.

The Basics

- Agile Manifesto: The foundation of agile working. Drafted in 2001 by seventeen software pioneers. The text is still surprisingly relevant and forms the core of how we look at value delivery today.
    - Source: agilemanifesto.org

- The Scrum Guide: The 'bible' of Scrum, written by founders Jeff Sutherland and Ken Schwaber. This contains the official rules of the framework.
    - Source: scrumguides.org

- Scrum.org: Where the Scrum Guide provides the rules, Scrum.org is the place for professional implementation. Founded by Ken Schwaber, this platform offers articles, videos, and blogs about the practice. This is also the place for anyone looking to obtain an official certificate (such as PSPO or PSM).
    - Source: scrum.org

- The 64% Rule: This figure comes from the well-known Chaos Report by the Standish Group. Their years of research into IT projects show that, on average, 64% of all features in custom software are rarely or never used. For an analyst, this is the source to keep asking the 'why question' and to prevent waste.

Interview Techniques

- Behavioral Interviewing: The technique of asking about specific past behavior is based on the STAR method (Situation, Task, Action, Result). It is the best way to replace assumptions with facts.

- 5 Whys: This technique comes from Sakichi Toyoda (the founder of Toyota). It is the simplest way to dig from a superficial symptom to the actual root of a problem.

- Focus on Emotions: Understanding a user's frustration or hope is a core part of Design Thinking. The Empathy Map, developed by Dave Gray (XPLANE), helps you systematically map out what a user truly experiences.

- Sketching a Future Scenario and Asking About Previous Solutions: These techniques stem from the psychology of Steve de Shazer and Insoo Kim Berg. It forces the conversation partner to step out of 'problem mode'.

- Imagine it Failed: This technique—where you pretend beforehand that a project has already flopped in order to uncover the risks—was conceived by psychologist Gary Klein. It is the way to address the 'elephant in the room'.

Techniques That Structure Your Work

- Flowcharts: The foundation for making processes visual was already laid in 1921 by Frank and Lillian Gilbreth. In the modern IT world, this has been further professionalized into the BPMN standard (Business Process Model and Notation). It helps you follow the 'logical path' of information without getting lost.

- MoSCoW Method: This way of prioritizing was developed by Dai Clegg during his work at Oracle UK. It has since become the standard for making choices that are understood by everyone.

- Personas: The concept of creating fictional users for human-centered design was popularized by Alan Cooper. It helps you stop thinking about 'the user' and start thinking about 'flesh and blood'.

- INVEST Criteria: Bill Wake came up with this acronym to test the quality of User Stories (our building blocks). If a chunk of information meets these six letters, you know it is ready for the development team.

My Personal Reading Tips

If you have caught the bug after reading this book, these are the books that have been on my own nightstand:

- "The Back of the Napkin" by Dan Roam. Because a drawing says more than a thousand minutes of meeting notes.

- "User Story Mapping" by Jeff Patton. The book if you want to learn how to make a large mountain of work visual without losing the overview.

- "Start with Why" by Simon Sinek. Although it is a management book, the core is essential for every analyst: always begin with the purpose, and only then move to the execution.

- "Scrum: The Art of Doing Twice the Work in Half the Time" by Jeff Sutherland. A smoothly written book that explains the philosophy behind short-cycle working.

About The Author



Michiel Lensink isn't a theorist watching from the sidelines but likes a hands-on approach to his work. He has been active in the world of IT and software development since the early 2000s.

He began his career as a developer but soon realized that success depends on more than just good code. Today, he fulfills various roles, ranging from Product Owner and Information Analyst to Scrum Master and Agile Coach.

This diversity of roles is a conscious choice. Michiel doesn't like to be put in a box; instead, he chooses whichever role is needed at that moment, driven by the strong conviction that this is where he can make the most difference.

Along the way, he has developed a special interest in the human side of software development. While processes and technology are important, ultimately, it's the people who have to do the work.

With his company Consultancy En Zo, he bridges the gap between 'business' and 'IT,' always with a down-to-earth approach and a healthy dislike for unnecessary paperwork.

Looking to bounce ideas around about your project or hire Michiel? Visit www.consultancyenzo.nl or send a message via LinkedIn.

Glad you read my book. I hope it provides you with valuable practical insights."

If you found it helpful, a small contribution would be greatly appreciated. You can donate via the QR code below.



Or buy a hard copy via www.consultancyenzo.nl