# From Projects to Product Teams

## How to Design a Product Team Organisation That Holds Up

Whitepaper, November 2026

# Executive Summary

A large number of IT organisations are moving away from projects and functional departments, towards product teams. The reasons are consistent: delivery is too slow, every change needs three departments, nobody owns an application from end to end, and the few people who understand the critical systems are involved in everything.

The direction is sound. The way the new organisation is usually designed is not. In most cases it is produced in one of three ways: from a capability assessment, from the current organisation chart, or from a workshop that starts with a blank sheet. All three produce a structure that looks reasonable on a slide and does not survive the first quarter of real work.

This paper explains what a product team organisation is, why five different types of teams are needed rather than one, and why a design fails when there is no frame underneath it. It then describes the approach we use ourselves to produce a first version of a target organisation. The approach asks for seven inputs, all of them facts the organisation already has. The model work happens outside the room, before anyone is asked anything. The workshop then has one job, which is to confirm or correct a concrete proposal instead of inventing one from nothing.

This paper describes the approach, not a service. It does not reproduce our models in full, but it explains what the method needs as input, what happens to that material, and what comes out of it.

# Why Organisations Move to Product Teams

The decision to reorganise rarely starts as a question of structure. It starts with a list of symptoms that most CIOs will recognise.

- Delivery takes longer than the business can accept, and the delay is almost never caused by a lack of effort. It is caused by waiting.
- A single change needs several departments, and each handover adds a queue.
- Applications have no clear owner. Projects deliver them, projects end, and what is left behind is maintained by whoever is available.
- A small number of people appear in every important decision. If one of them leaves, several systems become fragile at once.
- The business cannot name who decides the order of work for the systems it depends on.

A product team organisation is an answer to exactly these symptoms. A product team owns what it builds and it runs what it owns. It has one backlog, one owner and one roadmap. Work arrives in one place, one person decides the order, and the team is accountable for the result over time instead of until the end of a project.

That is the whole promise, and it is a good one. The difficulty is not in the promise. It is in drawing the boundaries between the teams.

# What Makes a Team a Product Team

The test is simple enough that anyone in your organisation can apply it from experience, without any theory: one backlog, one owner, one roadmap, and the team runs what it builds.

From that test follows the rule we use when a boundary is disputed in a room. If two pieces of work would be prioritised against each other by the same person, they belong in the same backlog. If they would never compete in the same priority conversation, they belong in different backlogs. This settles most boundary arguments faster than any model, because it refers to something people already do every week.

Two failure patterns are worth naming early, because they are the two most common outcomes of a design exercise that stops too soon. A team that has only a business label is a slogan, not a team. A team that is only a technical stack with a new name is the old infrastructure department, still handing out tickets.

# Five Types of Team, Not One

The most frequent design mistake we see is treating every team as the same kind of object, and calling all of them product teams. They are not the same. Five types exist, each with a different source for its boundary, a different consumer, and a different way of failing.

## Solution Team

Its boundary comes from a business value stream or an application the business can name. Its consumer is an end user or a business stakeholder. It owns very little of the platform and consumes almost all of it. It is permanent. Its failure mode is that it needs a ticket for everything, which means it is not autonomous, whatever the organisation chart says.

## Platform Team

Its boundary comes from a group of platform components with one coherent consumption model. Its consumer is other teams, not the business. This is the mirror image of a solution team: it owns outright what others consume. It is permanent. Its failure mode is a backlog driven by its own technology roadmap instead of by the teams it serves.

## Specialist Team

Its boundary comes from expertise that is scarce, or that carries enough risk that it has to be pooled and protected. Its consumer is many teams, on demand. It is permanent, but it should become smaller over time. Its failure mode is that it turns into an approval gate, which is the fastest way to remove the autonomy you were trying to create.

### Enablement Team

Its boundary comes from a skill gap, not from a system. It helps another team become able to do something itself. It owns nothing permanently, although it often builds the first version of something and then hands it over. It has an end date, agreed on the day it is created. Its failure mode is that it becomes permanent and quietly turns into a governance body.

### Capability Team

Its boundary comes from a reusable technical product that can be tested against a fixed set of criteria. Cost management is the clearest example. It is permanent for as long as those criteria hold. Its failure mode is confusion with a platform team.

The order in which these are designed matters more than the definitions. Solution teams come first, always, because they are the only teams with a consumer outside IT, which makes them the only teams that can be defined without reference to another team. Platform teams follow from what the solution teams need repeatedly and cannot serve themselves. Specialist, enablement and capability teams follow from what remains.

> If you design the platform teams before the solution teams, you will recreate your current infrastructure department under new names.

### A Note on the "Spotify Model"

This is not the "Spotify model", and the difference is worth stating early. The idea that every team can be fully autonomous, choosing its own tooling, its own standards and its own infrastructure, looks attractive on a slide and is expensive in practice. What it produces is chaos and costs that nobody controls: the same problem solved five times in five ways, five sets of tools to secure and to patch, and a (cloud) bill that no single person in the organisation can explain.

Autonomy is real only when something underneath it makes the standard path the cheapest path. Solution teams sit on top of platform teams, and they consume what those platform teams provide through a stable interface instead of building it again for themselves. That combination works, and it is the reason the five types exist rather than one. Autonomy without a platform underneath it is not autonomy, it is fragmentation.

## Why This Fails Without a Frame Underneath It

Three starting points are commonly used for this kind of design. Each one is useful for something, and none of them produces a working organisation on its own.

### Starting From a Capability Assessment

A capability assessment answers one question well: can this organisation do everything it needs to do. It is the right instrument for completeness, and it is the right language for a conversation with a

CIO and with the business. It cannot produce teams, and the reason is structural rather than a matter of effort. A capability is an accountability. A team is a queue of work with one person deciding the order. These are different objects. The same capability appears inside twenty teams at the same time, so capabilities do not group themselves into teams. A capability gap can point to a reusable product, to a governance decision, or to a practice that every team has to apply. It never points to a solution team or a platform team, and those two types make up most of the organisation.

### Starting From the Current Organisation Chart

This produces teams, which is more than the capability assessment manages, but it gives you no way of knowing whether anything is missing. It also carries the existing silos forward almost intact, because every current manager has a good reason for their own box to survive the exercise.

### Starting From a Blank Sheet in a Workshop

This is the most popular option and the least effective. A room asked to design an organisation from nothing produces one of two outcomes, and we have seen both many times. Either the room goes quiet, because the question is too large to answer out loud, or it produces today's organisation with new labels on the boxes, which is the one result the customer was trying to avoid.

We should be honest about our own history here. We have run those sessions ourselves, with good preparation and with the right people in the room, and we got the same two outcomes. That is why we changed the approach rather than trying harder at the same approach. The conclusion we drew is simple and it has held up since: people validate far better than they invent. A room given a concrete proposal will tell you within minutes exactly what is wrong with it.

> Bring a proposal to be shot at, never a blank sheet.

## The Two Models Underneath the Approach

The proposal is built with two models that 45 Degrees has developed and maintains. Nobody in the customer organisation needs to learn either of them, and neither of them appears on screen during a validation session. They are instruments for building the proposal and for checking it afterwards, and they stay in the background.

### The 45 Degrees IT Capability Framework

Version 4.4 describes forty-five capabilities in five domains: Strategy, People, Platform, Security and Operations, with Business Alignment and Innovation Management in the centre as the Value Loop. It answers the question of what the organisation must be able to do. It is written to be readable by the business, which is what makes it the right language for validating a design with a CIO and with business stakeholders.

### The 45 Degrees Platform Component Model

This model describes twelve planes and seventy-six components. Each component is documented against the same twelve topics, including its purpose, its business value, its interfaces, its consumption model, its governance, its dependencies, its risks and anti-patterns, its KPIs, where AI can be applied to it, and a maturity model. It answers a different question: what the platform must actually provide. This is the depth an organisation design needs below the level where a capability stops being actionable.

Both models are needed, and for different reasons. The capability framework gives completeness. The component model gives depth. Every team in a finished design has an address in both: what it is accountable for, stated in business language, and what it owns and consumes, stated in technical terms. A team with only the first is a slogan. A team with only the second is a silo. The two also have to be consistent with each other, and checking that consistency is a large part of the work that happens before anyone sees a proposal.

## The Seven Inputs the Method Needs

Seven inputs. Nothing else, and nothing to prepare. Every item is a fact about the organisation, in its own words, taken from material that already exists.

- The project list of the last twelve months, with one short line per project on what it delivers and for whom. If you run demand management, the demand list instead or in addition. The description matters more than the project name.
- The project list of the next twelve months, in the same format. Planned, funded, or seriously on the table.
- A ticket export of the last twelve months. The raw export, all of it, with no filtering and no cleanup. This usually takes ten minutes to produce and it is the richest single input we receive.
- The application list, with the business owner per application, from your CMDB.
- The current IT organisation, including external staff, with one line per person on what that person actually does. Not the job title. Two people with the same title often do completely different work.
- Ten recent examples of work getting stuck. One line each: what was late, and what it was waiting for.
- Five to ten names of the people everybody needs, and what people come to them for.

If collecting this takes more than a few hours, the request has been made too large, and that is a mistake on the asking side rather than a problem with the organisation. Any format works. A messy export tomorrow is worth more than a clean document in three weeks.

Three things the method deliberately does not ask for: an assessment of the organisation's own capabilities, an inventory of platform components, and any opinion about what the future

organisation should look like. Those belong to the model work, and the model work is not homework for the customer.

### What Each Input Reveals

The request is short, so it is fair to explain why so little is needed. The project and demand history shows who asks for work and about what, which is where solution teams come from. The ticket export shows which systems always appear together, and which teams raise tickets against which other teams, which is where platform teams come from. The application list with business owners shows the landscape and the business side of each future backlog owner. The one line per person shows what the organisation actually does, as opposed to what it is called. The stuck work shows which handovers are in the wrong place today, and it produces the business case at the same time as the design. The names everybody needs point to specialist and enablement teams, and to your key person risks.

### Why the Next Twelve Months Matters

Six of the seven inputs describe the organisation as it is and as it was. A design built on history alone reproduces the past with better labels. Most organisations want a new structure because of what is coming, not because of what happened. The forward list shows which teams have to exist although they have no history at all, where the organisation will grow and where it will shrink, and where scarce skills will be needed next. It also reveals the real strategy, because what is funded for next year is what the business actually intends, whatever the strategy presentation says.

> Past evidence sizes the teams. Future evidence shapes them.

### What It Means When an Input Does Not Exist

Every one of the seven inputs is something a reasonably run IT organisation already has in some form. When one of them cannot be delivered, that absence is information in itself, and it is far more useful to say so plainly than to spend two weeks building it. A missing project list says something about portfolio management. A missing business owner per application says that ownership sits inside IT rather than with the business, which is a design constraint from day one. None of this is treated as a failing. It is recorded, and it returns in the second iteration as a conclusion the organisation reaches for itself.

## How the First Proposal Is Built

The seven answers are mapped onto the two models before anyone meets. This mapping is preparation for the workshop, not a deliverable in its own right, and it is the part of the work the room never sees.

What the room does see is one short document with three parts.

- One page per candidate team. Fixed format: the name people will use for it, one sentence in business language on why it exists, which of the five types it is, the backlog owner by name, what the team owns, what it consumes from other teams, and the one thing we could not decide from your input.
- A list of open questions, which are the points where your material did not give us enough to make the call.
- A list of everything that has no owner, and everything that has two owners.

If a team page does not fit on one page, the team is not understood well enough yet, and it is either split or the missing part is written down as the open question. The page talks about applications, people and work. It does not talk about models.

That third list, the items with no owner and the items with two owners, is usually the most valuable output of the whole exercise. An item with two owners is where a current silo boundary is hiding. An item with no owner is either a real gap or a deliberate decision not to have it, and both are acceptable answers as long as the decision is written down.

## The Workshop Validates, It Does Not Create

The session has one job: confirm or correct the proposal. Per team, three questions only.

- Is this one backlog with one owner?
- Is that owner the right person, by name?
- Is anything missing from what this team owns?

Anyone in the room can answer these without preparation and without knowing anything about the models. That is the test for whether a validation question is well formed. If a question needs a model explained first, it is the wrong question for this session.

After the team pages there is one real discussion, and only one: the items with no owner and the items with two owners. That is where the genuine disagreement sits, and it deserves the time. Everything else is confirmation.

One point of realism about the agenda. If the proposal contains twenty teams, there is no point walking through all twenty. Take the ten that matter and say so. A session that covers ten teams properly is worth more than one that covers twenty superficially, and the pattern is what the organisation needs to learn in any case.

## The Second Iteration: Is Anything Missing

Corrections are folded in within a few days, and then the question changes. The first iteration asks what the teams are and who owns what. The second asks whether anything has been forgotten, and this is where the capability framework comes in.

The forty-five capabilities are walked against the proposed organisation, and each one is given a verdict: owned by a specific team, carried inside the teams for themselves, consumed from somewhere else, or part of the framework around the teams. Whatever lands nowhere is a real gap and goes onto the roadmap, with a priority.

What lands in the framework column is the governance design: demand intake, portfolio decisions, architecture guardrails, supplier management, career and skills paths. It is worth noting that this is not invented separately and not copied from a reference model. It is what remains once the teams have taken what they carry. That is also the concrete content of the phrase autonomy within a framework, which is otherwise difficult to make real.

Running this completeness check first, before concrete teams exist, is what makes people lose the thread. The capability framework can tell you whether your organisation is complete. It cannot tell you what the teams are, so it is the wrong instrument to start with and the right instrument to finish with.

## What This Is Not

The approach has limits, and they are worth stating plainly.

- It is not a re-organisation plan. It produces teams, boundaries, ownership and gaps. Decisions about people, headcount, reporting lines and timing sit elsewhere, with the HR and works council process.
- It is not a single pass. Two or three short iterations work better than one long study, and each pass is a proposal followed by a correction.
- It is not a repair for a weak foundation. A product team organisation exposes weak engineering practice, fragmented service management, and a platform that is not self service. If a solution team cannot do its normal work without a person in another team, its autonomy exists only on paper.
- It does not move the decision. The first proposal comes from outside the organisation, on purpose, so that it can be argued with. Every boundary and every named owner in the final version belongs to the organisation itself.

## What Makes This Work in Practice

- Be clear about whether the goal is a design or a discussion. This method produces a design, and it expects to be corrected.
- Collect the seven inputs as they are, with no cleanup, and say plainly what does not exist.
- Pick the two delays from the last year that cost the most. They anchor the whole conversation in something everyone recognises.

- Put the people who can confirm a backlog owner by name in the validation session. A team whose owner cannot be named in the room exists on paper only.
- Plan for two iterations, a few weeks apart, rather than one large workshop.
- Expect the gaps. The findings are usually the most useful part of the exercise, and they are rarely comfortable.

## Conclusion

The reason most product team designs disappoint is not that product teams are the wrong model. It is that the design was produced in the wrong order, by the wrong method, and often by the people with the strongest interest in the current structure. A capability assessment on its own gives you completeness and no teams. An organisation chart on its own gives you teams and no completeness. A blank sheet in a workshop gives you silence, or the organisation you already have with new names on the boxes.

The alternative is not more analysis. It is a different division of labour. The organisation provides facts about itself, which costs a few hours. Those facts are mapped onto a capability framework and a platform component model, and what comes back is a concrete proposal with named teams, named owners and a clear list of what has no owner and what has two. The people in the room then do the thing people are genuinely good at, which is saying precisely where that proposal is wrong.

This is the approach we apply ourselves, and the two models behind it are our own. Both are described here only as far as a reader needs in order to judge whether the method is sound. We are happy to discuss it in more detail. Contact us at info@45degrees.be.