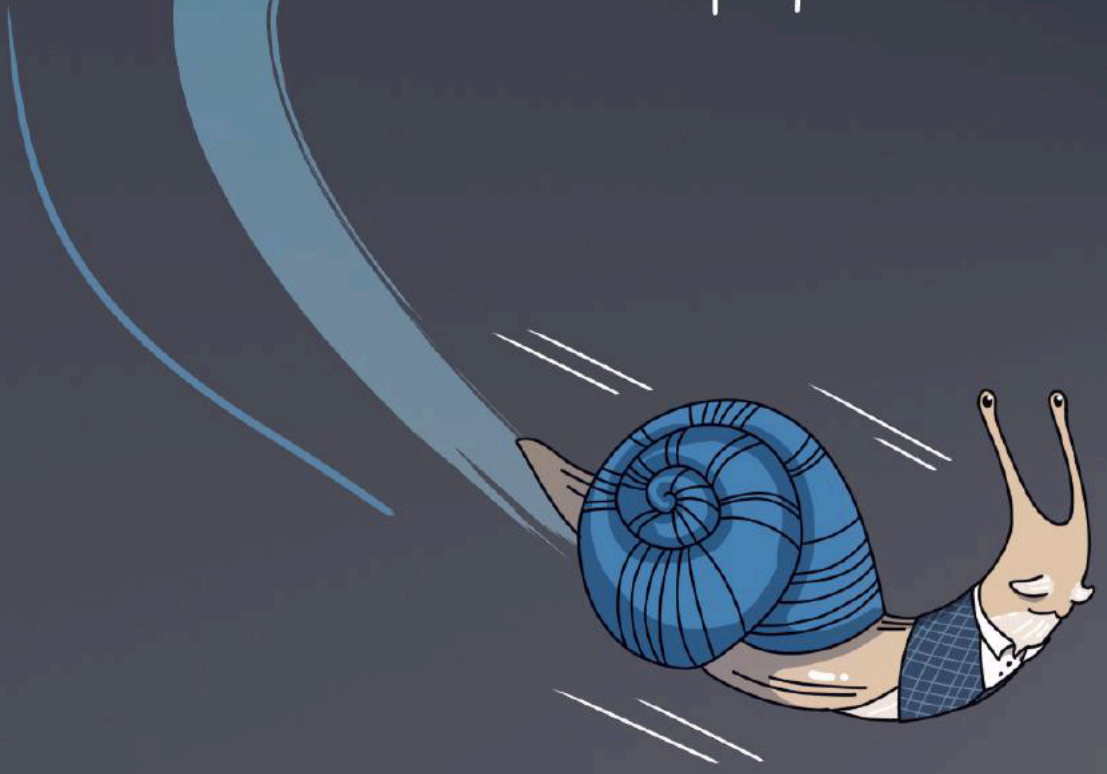
What makes an organization slow and how to fix it

BY

Jan Grape



SNOWDROP

What makes an organization slow and how to fix it

by
Jan Grape

Version: 1.0.1

Publisher: Jan Grape
Cover art: Luna Sakura

Contents

Acknowledgements

I'd like to thank my colleagues at Snowdrop for cheering me on and giving feedback. In particular, Mattias Skarin, Lennart Boklund, Jimmy Janlén, and Marcus Lagré suggested changes, additions, and improvements to this little book.

Introduction

This book is about what holds organizations (not just single teams) back from creating value faster.

Being fast is a good thing since:

- It lessens the time our end-users have to wait for new stuff that can help make their life better
- It accelerates return on investment (ROI) for us since we can start charging our customers sooner (or collecting other types of value).
- This increases feedback so we know better if we are doing the right thing or need to change course.

After 15 years of helping organizations improve, my clients' number one question is: "*Can we get more out of the people and organization we already have?*" The answer is almost always: "*Yes, you can.*"

It's not about cracking the whip and making people work harder under pressure and longer hours. That's not sustainable and, in my opinion, inhumane. On the contrary, it's about making it **easy to be fast without extra effort,** and I know a little about that, hence this book.

Being fast is not the only important thing. Even more important is creating products that serve people and your business well. Choosing what to build and what not to build in a product discovery process is a topic for a future write-up. Today, we start with speed.

The chapter "**Don't start at the wrong end**" discusses where to start to increase your organization's speed. Bottlenecks and dependencies might stand in the way, but if people don't care, you won't get far despite your best efforts.

In the chapter **"Dependencies kill flow,"** we explore perhaps the main reason for an organization's slowness. Dependencies between teams and units mean synchronization in time (one delivers and one receives), and synchronization almost always leads to one party waiting for the other, which adds to lead time, i.e, the time customers have to wait to experience your work.

"Creating organizational focus" looks at the most common solution to the dependency problem: having teams work on different things and releasing them independently. It sounds excellent, but the consequence is that you spread your organization thin, reduce collaboration, and have competing priorities.

In the chapter **"Don't scale with underperforming teams,"** we look at the building blocks of your organization. I will introduce some key points for high-performing agile teams, but this subject really requires a whole book on its own, which may come in the future.

Finally, in the chapter **"Continuous improvement on an organizational level,"** we look at how to make the state of being a fast organization permanent. This means that the leadership has to build an environment where people can perform and have a system for identifying things that suck and fixing them when they can't be addressed on the team level.

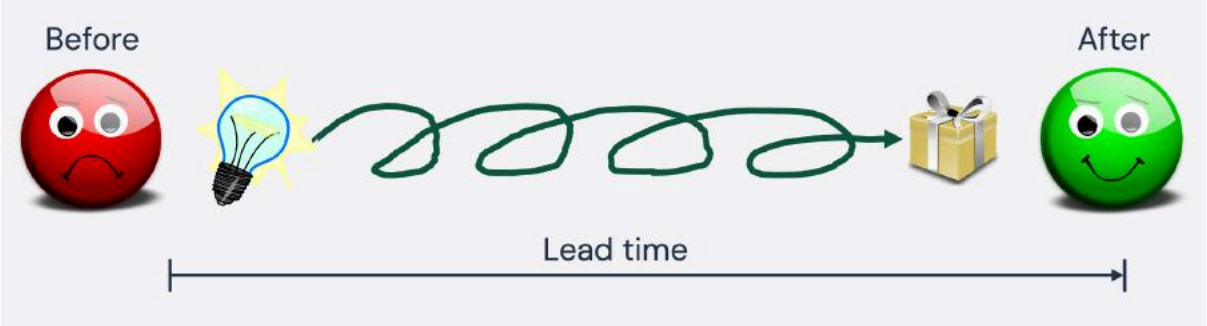
I hope you'll join me on this little journey.

Don’t start at the wrong end

It may sound counterintuitive, but when trying to become faster, the first thing you should do (IMHO) is not try to be quicker by adding more people or any other means. Three steps come before.

Measure your speed

The true metric of organizational speed is lead time. This is the period from when you decide to build a thing, solve somebody’s problem, enhance your product, etc., until your end-users have a new user experience in their hands.

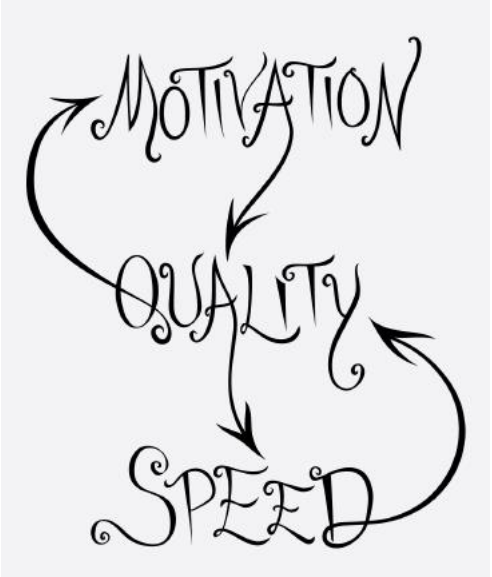


Lead time starts ticking when you decide to solve someone’s problem and stops when their (work) life improves, not when your gadget is ready for delivery.

The easiest way to get a baseline measurement is to take a few of your recent deliveries and backtrack to when your organization decided not to say “no” to them. That can be pretty far back in time, sometimes years, often months, but ever since then, somebody has been waiting for that request to be fulfilled.

Make sure your people are motivated

Motivated people do a better job; they care and are willing to jump through some hoops to get what’s needed done. Also, their well-being is higher, and they score higher in job satisfaction surveys.



That's great! Let's motivate everyone 🙂 But how do you do it?

I have a simple trick: Motivation comes on its own once the biggest de-motivators are gone!

I simply ask people the question, "***What sucks the most in your professional life right now?***"

It's a brutal question, but I have found that it often generates an honest answer. Ask enough people this question, and you'll usually see some patterns emerge.

Identify the worst de-motivator (whatever it is), start fixing it transparently (show people you are taking their frustrations seriously), wait a couple of weeks, and ask individuals the same question again.

Why is this the first step? If people are demotivated, doing quality work is hard (they don't care), so the next step becomes much more complicated.

Speed without quality is useless

Quality work comes in two flavors: Doing the right work (building the right thing) and doing the work right (without defects).

If you increase speed without having quality in place first, you'll just produce crap at a higher rate, and that's no good.

As I said in the intro to this book: ”Choosing what to build and what not to build in a product discovery process is a topic for a future write-up.” Therefore, let’s focus on the defect part for now.

The right place to measure true quality is out there in production. Measure the influx of defects, customer service requests, questions, critical expressions on social media, incidents, etc. Ideally, all of these should be zero, but they never are.

Surprisingly, these feedback loops are often weak in many companies. Commonly, the feedback doesn’t go all the way back to the decision-makers and creators of your product and services. It stays somewhere in the customer-facing part of the organization and becomes statistics.

Ensure there is transparency within the organization, set goals for every one of these feedback loops, and start improving.

Finally, accelerate

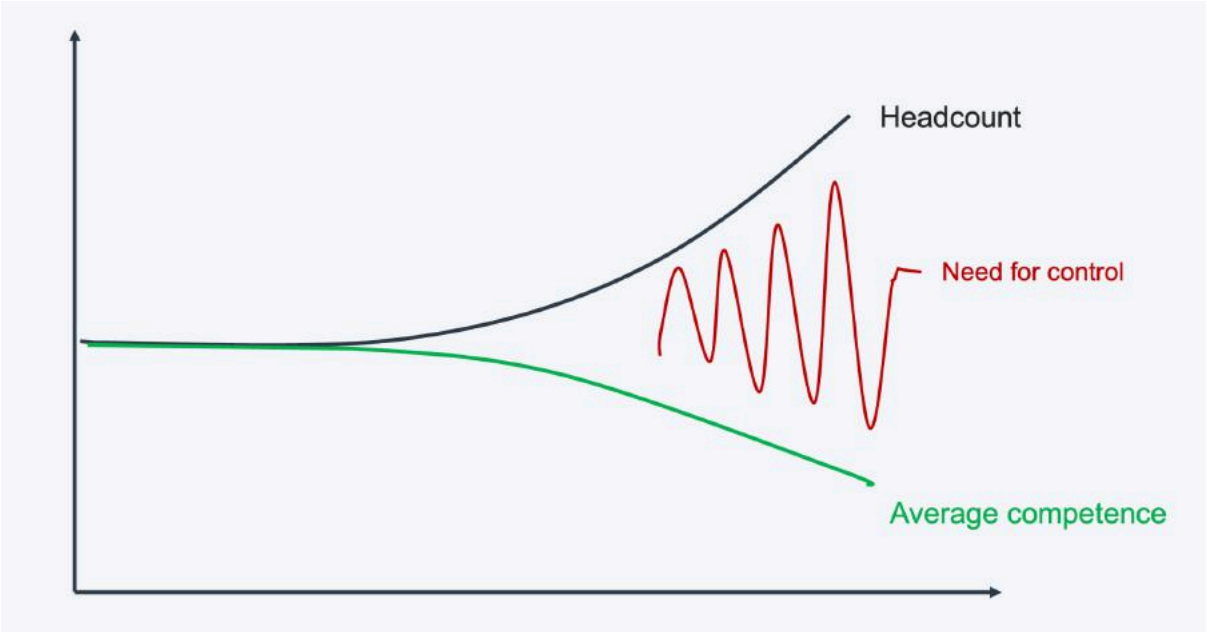
Once you have motivated individuals and teams that do quality work, increasing speed is much easier.

Instead of rushing to hire more people, ask yourself: Are my teams already performing at the peak of their capability, or are things holding them back?

Why not hire more people to increase capacity?

You might have to, but doing so has three significant drawbacks:

It’s a slow lever to move. Not only is it hard to find the right people, and attracting talent takes time, but when you do find them, they have to give notice at their current employer. Depending on where you are in the world, the notice period varies from days to six months. The most common notice period in Sweden, where I live, is three months for a senior-level employee.



The growth dilemma. Initially, everybody on the staff has the full picture. As headcount grows, fewer people know the whole thing; they just know their piece of the puzzle.

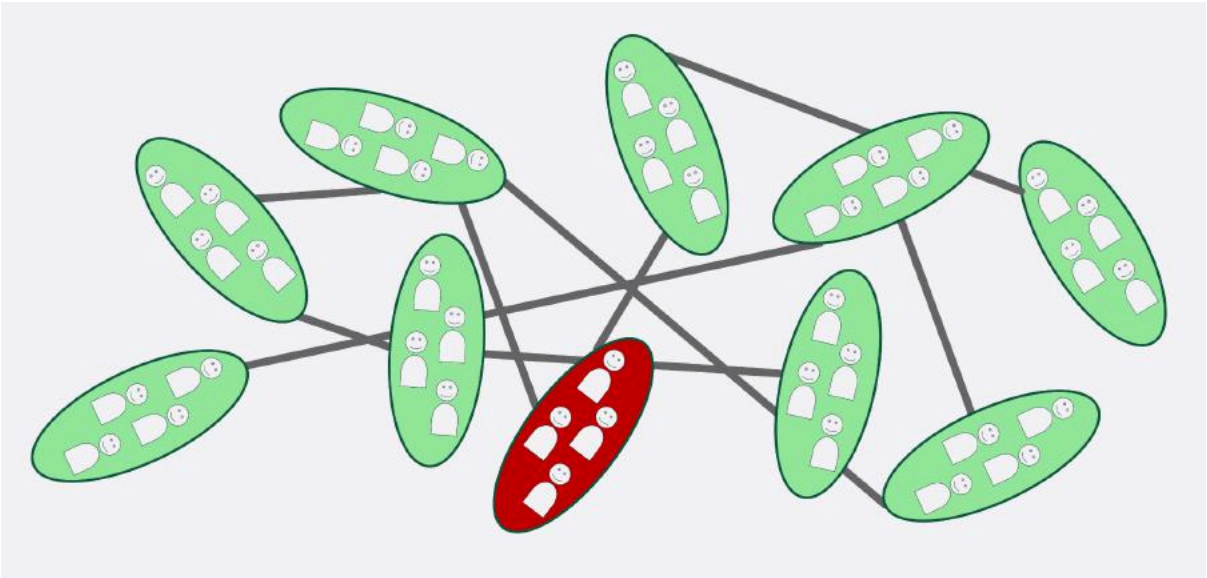
Adding people **dilutes the company's competence** regarding how your business, products, organization, architecture, and source code work. You'll have a bunch of newbies who need onboarding, which takes time and energy from the existing people in the company, slowing everyone down.

Not to mention team dynamics. Every time you add or remove people from a team, they have to reform and find everyone's place in the new constellation. This also temporarily reduces productivity.

The last and most obvious drawback is cost. **Staff costs money**, and if you can avoid adding more staff to increase capacity and speed, that's a good thing.

The following chapters in this book are about becoming faster without hiring more people.

Dependencies kill flow



The number of internal dependencies determines the maximum speed of an organization. If one team is late or slow to respond, other teams will suffer, too, and the problem can spread to everyone in a domino effect.

If a group of collaborating teams has dependencies, the result can't be produced sooner than the slowest team within the group. This is a fragile situation because any team can run into trouble for whatever reason and trigger a delay.

Will the fastest climber summit first?

A dependency means that one team delivers something (code, knowledge, services) that the other needs. An exchange like that requires some form of synchronization between the two, and synchronization (almost) always means that one party has to wait for the other, making one of them slower than it would have been otherwise.

Are dependencies bad? Should we remove all of them? No!

Some dependencies are necessary. Without them, applications and workflows wouldn't function properly. For example, a mobile app often has to exchange information with a server backend. Any information exchange leads to a (usually desired) dependency. Another example is a workflow dependency, like the dependency between a bank and its customers. To deposit and withdraw money from an account, there has to be a value and information exchange between the customer and the bank, hence a dependency.

These were two examples of desirable, valuable dependencies. What we don't want are dependencies that cause us grief.

Four types of dependencies

Before we figure out how to find the harmful dependencies, we should look at the different types of dependencies. I like to boil them down to four different categories:

- Missing skills or knowledge
- Not having the mandate
- Architectural and technical
- Process and Scheduling

Missing skills or knowledge is one of the most common reasons for inter-team dependencies:

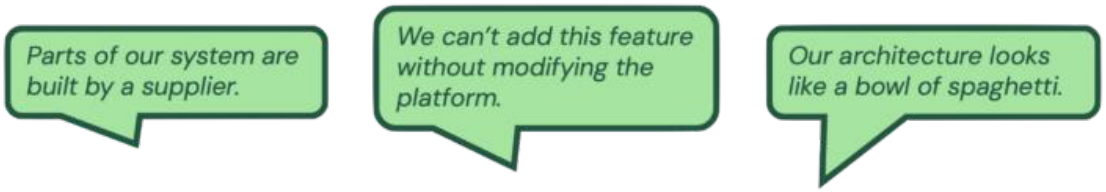
Here, teams should strive to become even more cross-functional and adopt skills and knowledge from others until they can solve their problems independently. It's perfectly fine to ask for help, but that should not always happen for the same things. If you need the skills and knowledge, solve it through training or recruiting for the team.

Not having the mandate is often coupled with the missing skills or knowledge, but not always.

Often, teams are responsible for a piece of code that nobody outside the team can modify independently. Collective code ownership isn't possible because other teams are not trusted to have the skills and knowledge to make safe changes in code created by the originating team.

Resolving the reason for the lack of mandate to make the necessary changes or operations is the way to remedy this situation. It could be a trust issue, a process issue, a knowledge issue, or something else. Base your organization on trust, transparency, and competence.

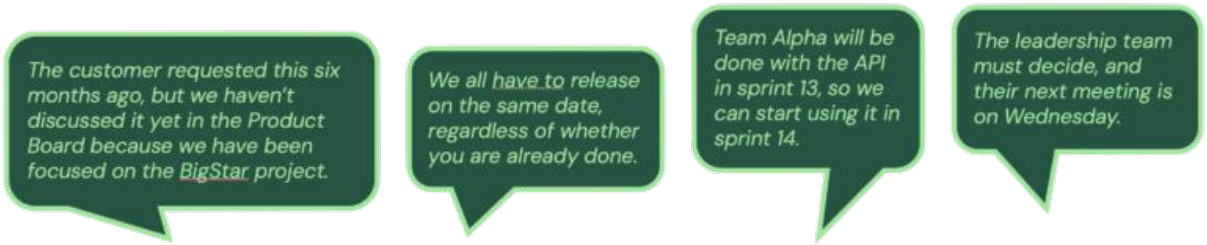
Architectural and technical reasons for dependencies are perhaps the first thing we engineers think of when talking about dependencies:



Problematic dependencies are often well-known and understood, but fixing them is another story. Fixing technical dependencies sometimes involves changing the application's fundamental structures, and postponing such changes is too easy.

When refactoring an architecture (yes, you have to do it bit by bit), focus on creating a structure optimized for high development speed, modifiability, and the fact that multiple teams will work in parallel in the same code base.

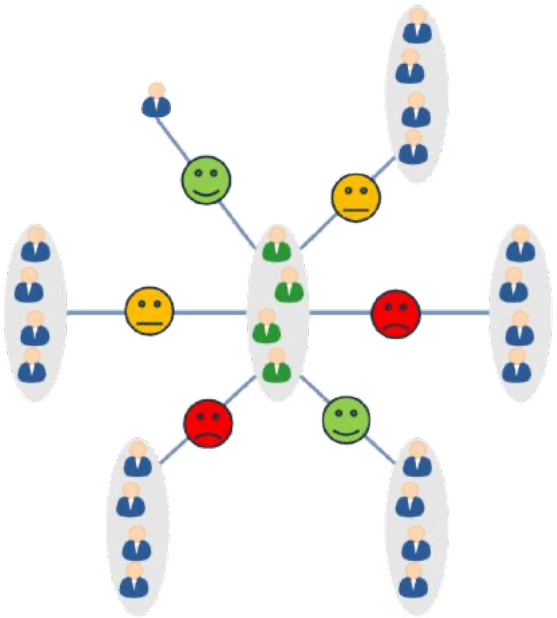
Dependencies on **Process and Scheduling** often cause delays where teams are not allowed to move on:



There is a lot that can be done in this area. Wasted time is often not visible because things are left lying around waiting for some condition to be fulfilled. Follow the flow of your deliveries and identify waiting times. Ask yourself, "Can we make this move along quicker?"

Identify and fix the harmful ones

Let every team in your organization create a dependency map to identify painful dependencies. Only those require attention. Dependencies that a team doesn't find troublesome don't have to be fixed immediately.



A dependency map for the "green" team

A dependency map is created as follows:

1. Put your team in the center
2. Identify your external collaborators
    - Other teams
    - Receivers of our work
    - Customers & Users
    - Stakeholders
3. Rate each relation
    - Use a smiley
4. Work on improving the sad ones
    - Talk to them
    - Show your dependency map
    - Address the underlying reason that makes the dependency painful.

It's essential to let teams start working on their most painful dependencies themselves, and it's equally important that they have the backing and support of leadership when doing so. It should be at the top of management's priority list to support their teams in improving their most painful dependencies.

This bottom-up approach creates engagement and promises to address dependencies that matter to each team.

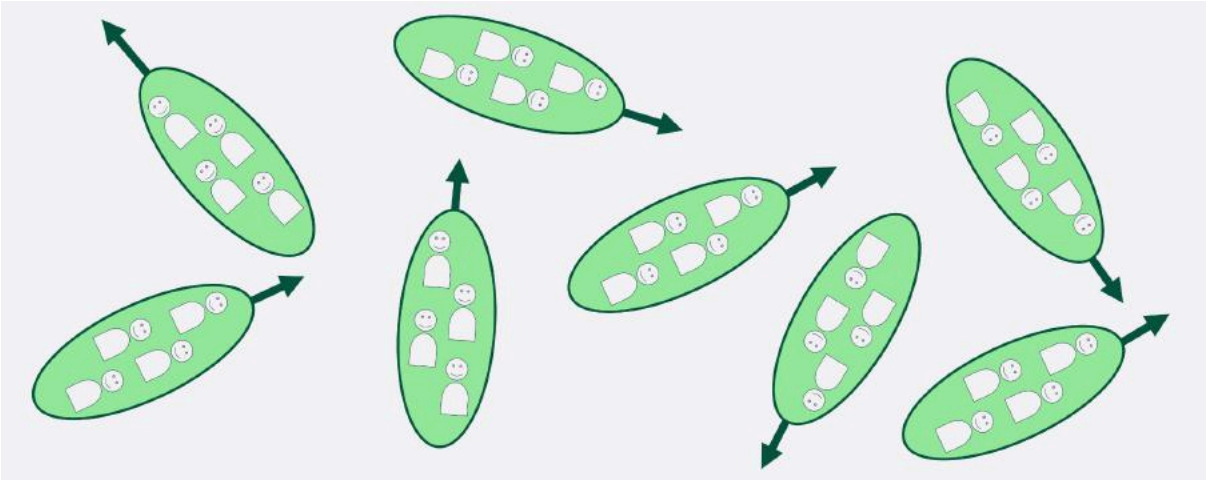
Is it a local optimization? Possibly, but that hasn't proven to be an issue in my experience. What's more important is starting to improve the situation. In one organization, several teams pointed to the same dependency as particularly problematic. With this knowledge, the problem could be addressed on an organizational level.

Now, go and make some dependency maps with your teams 🙂

Creating organizational focus

One of the most common solutions to the dependency problem in an organization is having teams work on different things and releasing them independently. It sounds excellent, but the risk is that you spread your organization thin, reduce collaboration, and have competing priorities.

Lack of alignment is a lack of focus



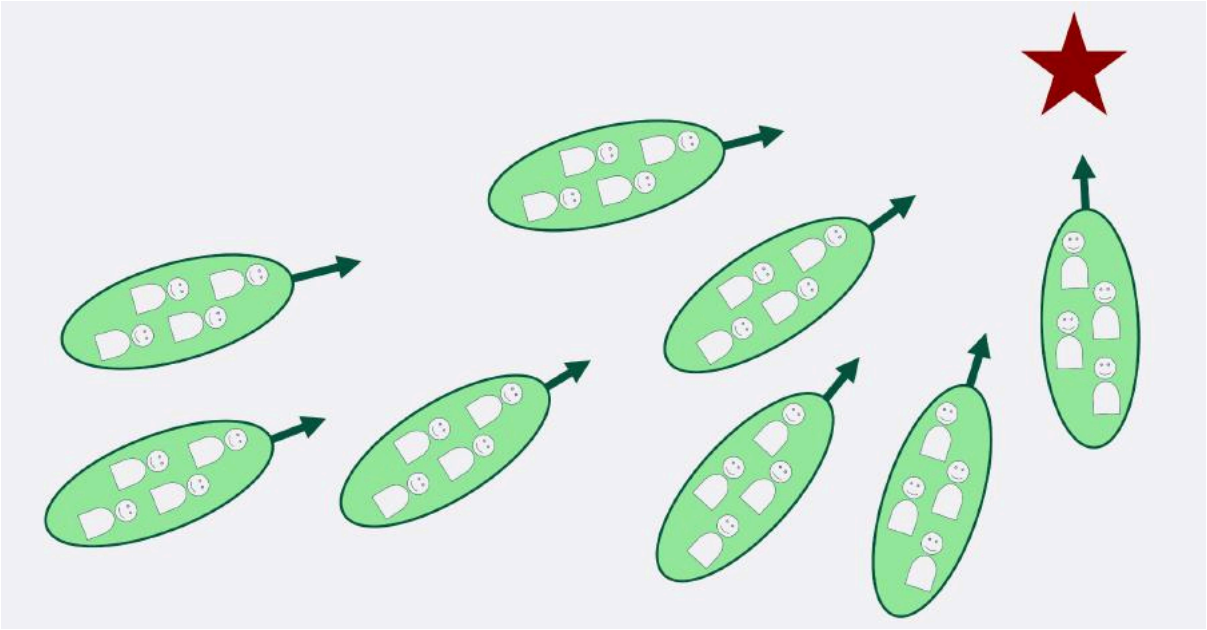
High autonomy & low alignment may lead to poor execution as a whole

The company risks not going anywhere without a common goal for the teams. At least not intentionally.

How to create alignment

Alignment across self-organizing teams is usually created by:

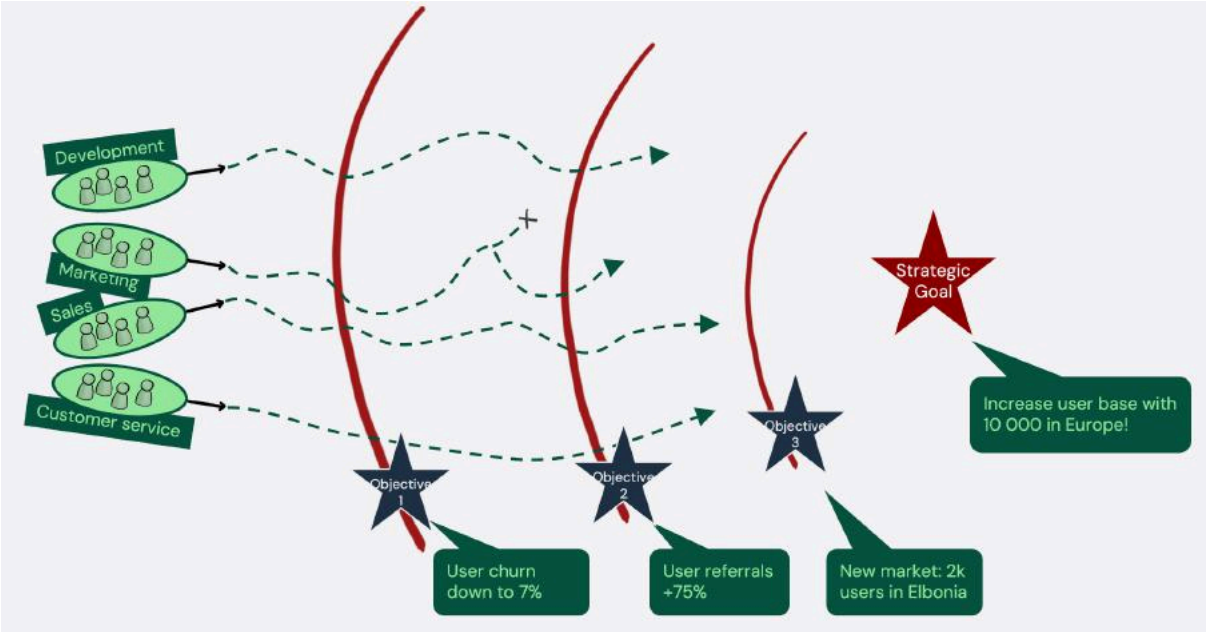
- Sharing goals and objectives
- Sharing priorities
- Sharing rhythm



Alignment is created by sharing goals across teams

Sharing goals also means sharing priorities. If you also share a planning rhythm (e.g., weekly, monthly, or quarterly), the teams can influence each other's plans for smoother execution.

Note that **teams that work towards the same goals and objectives don't do the same things**. Every team has a mission and purpose to fulfill, and what they deliver depends on that. Given their mission and purpose, they figure out the answer to the question: ***"What should we do to contribute to our common goal/objective?"***



Working towards the same objectives and goals, but contributing with different things

In the example above, the development, marketing, sales, and customer service teams all do different things to reach the common goal and objectives. For example, to reduce churn to 7%, the development team might add a desired feature, marketing works on strengthening the brand, sales introduces a better price model, and customer service implements a win-back team to reclaim lost customers.

Focusing the organization

Here are some areas to help create organizational focus:

1. Have strategic goals
2. Limit the number of goals
3. Goals should create a direction
4. Select a strategy
5. Rank your objectives
6. Limit the organizational WIP

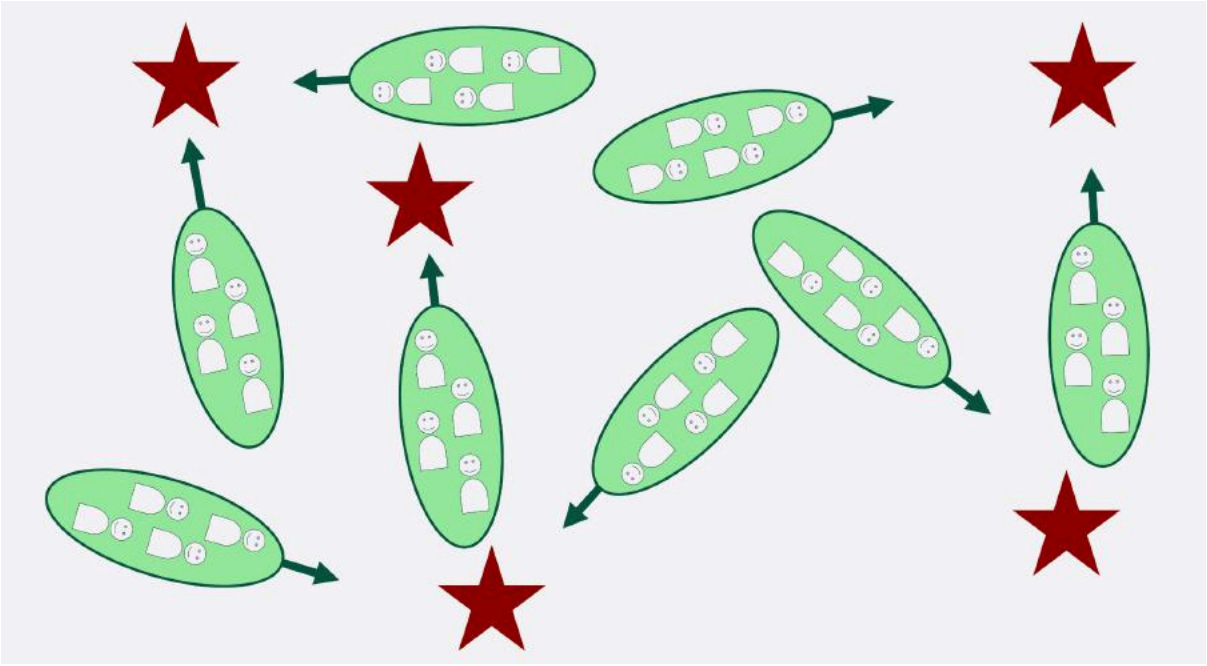
Have strategic goals

It is uncommon for mature companies to lack strategic goals, but it is very common for startups. Startups focus on getting their product idea out there, chasing cash flow and investors, and doing almost anything to achieve that. They tend to operate in an opportunistic and short-sighted way.

Luckily, startups are usually small, so organizational focus can easily be achieved across a handful of employees. However, when they scale up, strategic goals become essential for many reasons.

Limit the number of goals

The record number of strategic goals I have encountered at the top level in a company is 12(!). It's almost impossible to pursue and reach all of them. In my experience, most companies I have worked with settle on two to three strategic goals that matter.



Too many goals unfocus the organization

Too many goals unfocus the organization. It becomes up to every team to choose which one they think is the most important one to pursue.

Having a large number of strategic goals is often a clear sign that the leadership team is struggling to agree on what truly matters. By allowing many strategic goals, they avoid conflicts and discussions that are important to have.

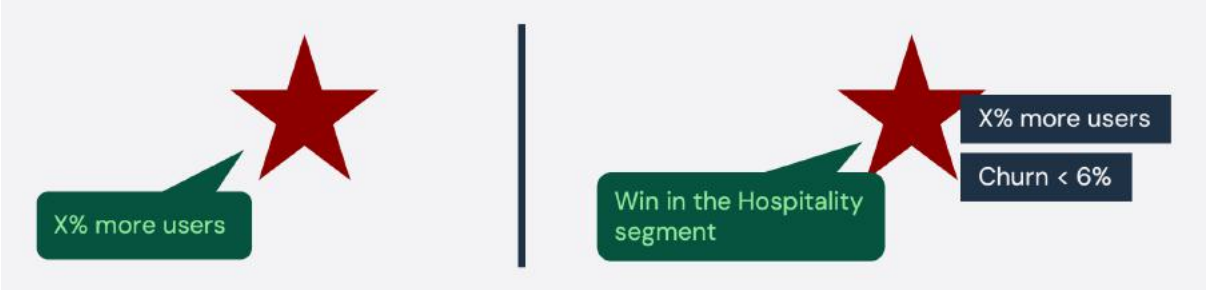
Goals should create a direction

Having a number alone (a KPI) to achieve as a goal doesn't provide a clear direction, opening up different interpretations of how to meet the target.

One company in the payment segment aimed to increase its customer base by X%. That was the KPI. It sounds straightforward enough, but a significant misalignment was revealed when the sales organization pursued new customers in the hospitality segment while the product development department built features for the retail segment. The salespeople never received the features they needed to meet their targets, and the new features that were developed were never used to attract more retailers.

Another company providing a service over the internet focused on reaching a specific number of daily active users (DAU). The CEO communicated this number clearly and repeatedly, but it was just a number to reach among several other KPIs. Until someone asked why this DAU number was so important? The answer was swift: "So we can beat competitor A!!". That made a difference. All the numbers to target now had a context and purpose, giving the company a direction and focus.

A goal (and an objective) describes a future state, a market position, a capability we don't currently have, or something else necessary for the company to thrive. The metrics (KPIs) only indicate whether we are moving in the desired direction.



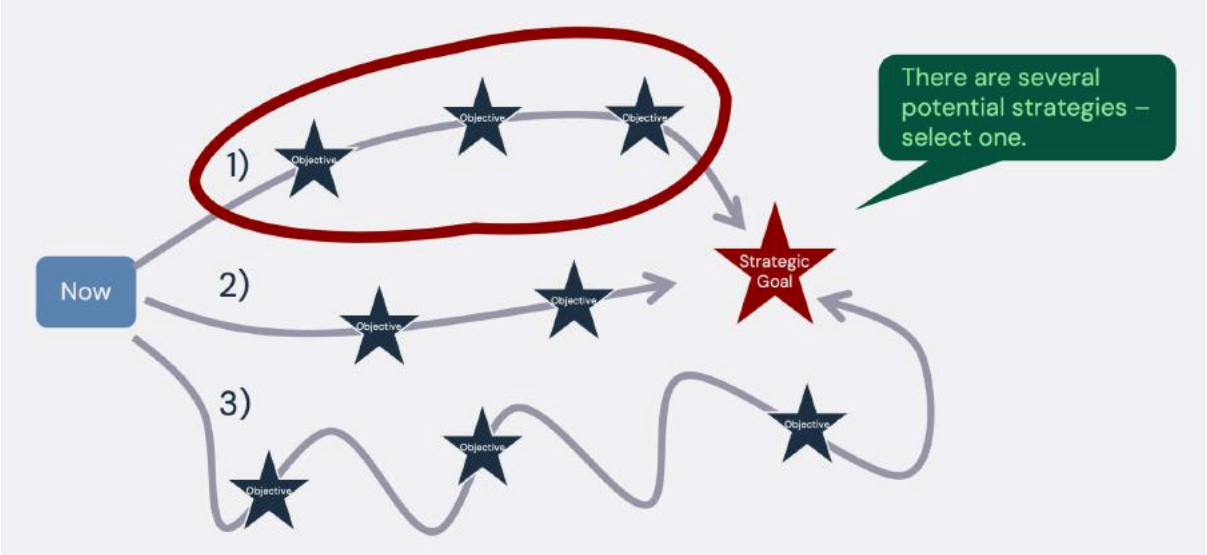
To the right: Metrics are used to measure progress towards the actual goal.

This is why OKRs (objectives and key results) are popular nowadays. The Objective part describes in words what we are striving for and why, while the key result part explains how we can measure our progress in that direction.

The trouble with OKRs is that they are just goals. To provide focus and drive, you need to decide what really matters; in short, you need a strategy.

Select a strategy

Strategic goals can't usually be achieved in one go. Therefore, it's helpful to break them down into objectives that eventually lead to the goal.



Select a strategy on the top level that explains the path towards the strategic goal (e.g., a desired market position)

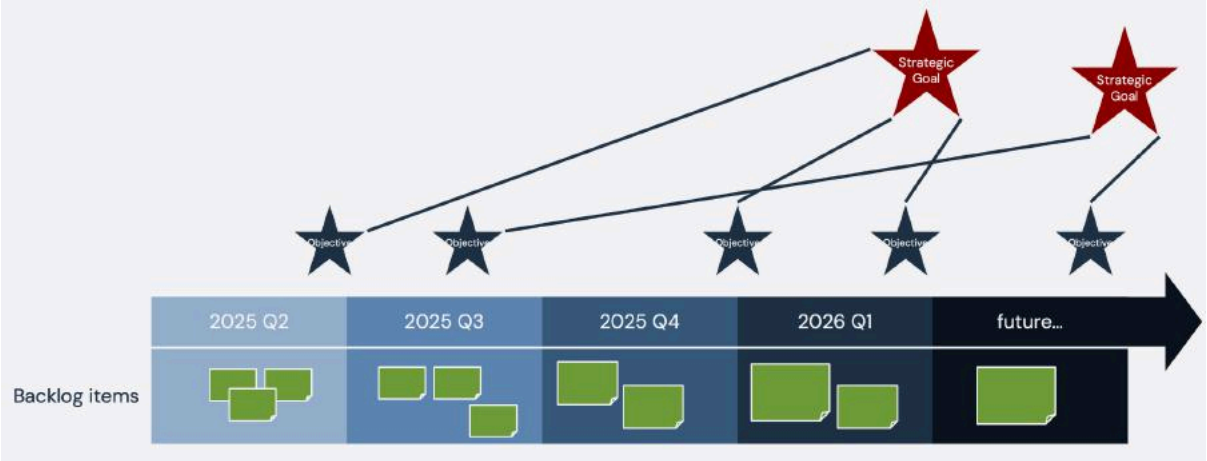
Your strategy can be described as the sequence of objectives you choose and the order in which they are selected. An objective in this sense is something achievable within a quarter to a year.

Many companies communicate internally only the strategic goals with emphasis and leave out the common path to get there. Although all parts of the company strive for the same ultimate goal, they plan different routes and risk having conflicting objectives along the way.

Rank your objectives

Having more than one strategic goal is usual, and having multiple objectives (as a result of your strategies) is equally expected.

With limited resources (number of teams), it's helpful to signal which objectives are in focus every quarter. This can be done with a priority list or, my preferred method, a timeline.



A roadmap. Objectives form the backbone. Deliverables (backlog items) can always be replaced as long as the objective is reached.

In the example above, the strategic goals are supported by the objectives. The timeline describes which objectives are in focus at any given time. This sequence can form the backbone of a roadmap based on objectives rather than deliveries.

Having a limited number of objectives in focus at any given time makes it easier for teams, as they are not constantly spread thin on goals and objectives. This also helps collaboration, as their neighboring teams share the same priorities.

Limit the organizational WIP

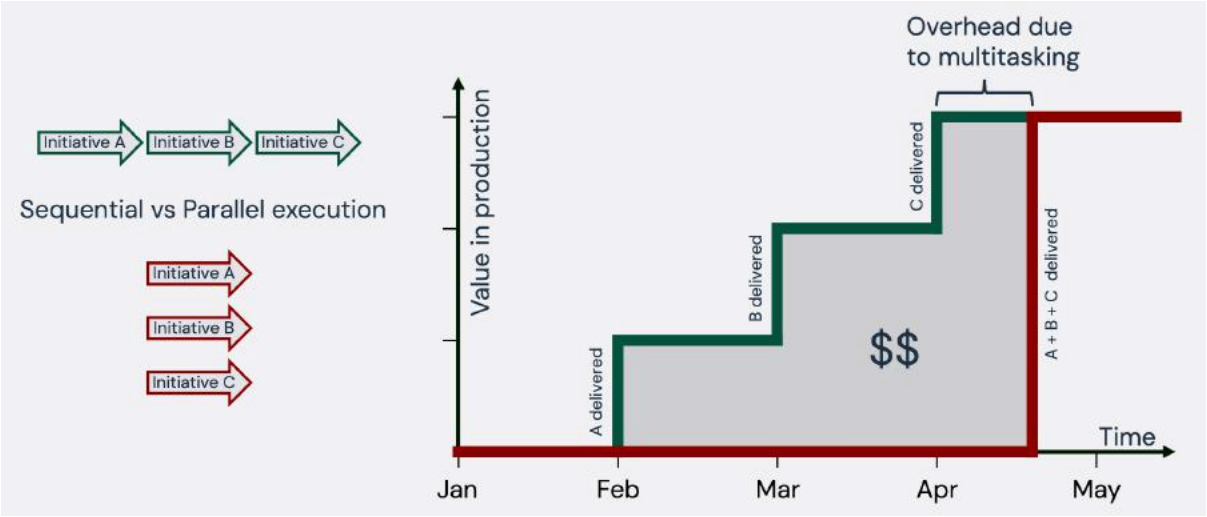
Limiting the number of strategic goals and focusing on only a few objectives at a time should prevent having too many initiatives with deliveries going on simultaneously.

If you make your initiatives (deliveries) small enough, you don't have to run them in parallel, and you can thereby accelerate your return on investment (ROI).

Imagine this scenario: you have one team and three initiatives to deliver. If the team works on one initiative full-time, it will take the team one month to build it.

Compare the two situations: the team works on all three initiatives in parallel vs. the team working on one at a time.

Which option will earn you the most money and provide the biggest customer satisfaction?



Read chapter 5 in Don Reinertsen's book The Principles of Product Development Flow to learn more about the advantages of smaller batch sizes.

The green curve represents time-to-value optimization, while the red curve represents resource optimization (always have some initiative to work on).

In the first case, the first customer will receive initiative A after one month, and no customer will have to wait longer for their delivery than in the second case. Starting several initiatives early doesn't automatically mean finishing early.

The example above is a simplified case, but it illustrates that having too many initiatives in flight will hurt earnings, customer satisfaction, and lead time (to value).

Summary

Having alignment is the foundation for focusing an organization on the most important things for the business. Alignment is built through:

- Shared goals and priorities
- Shared planning rhythms (e.g., weekly, monthly)
- Each team contributes differently based on its mission

To maintain and further increase organizational focus, organizations should:

1. **Have strategic goals** – Essential, especially for growing startups.
2. **Limit the number of strategic goals** – Too many dilute focus; 2–3 is optimal.
3. **Ensure goals create direction** – Goals should describe a desired future, not just KPIs.
4. **Select a strategy** – Break goals into prioritized, time-bound objectives.
5. **Rank objectives** – Use timelines or priority lists to highlight what matters now.
6. **Limit organizational work in progress (WIP)** – Too many simultaneous initiatives delay outcomes and reduce ROI.

Focusing on fewer, strategically aligned goals and limiting active initiatives improves collaboration, speeds up delivery, and boosts customer satisfaction.

Don’t scale with underperforming teams

There is too much to do and too little being delivered. Let’s hire some more people to increase our capacity. After all, the teams have been saying: “We need at least two more developers to keep up.”

This seems to make sense; if you are overwhelmed with work, get some extra hands to help out. However, **adding more people to your organization should be the last resort**. First, make sure you do the most with what you already have.

Should we add another truck or fix the ones we already have to increase capacity?

In previous chapters, we have covered the environment the teams execute within: ”Dependencies kill flow” and ”Creating organizational focus.” In this chapter, we look at what’s going on inside the teams.

This is important because optimizing your organization, for example, by applying SAFe, LeSS, the Spotify Model, or the most recent Product Operating Model, doesn’t help if the teams themselves have a long way to go in delivering fast with quality. It just adds complexity and bureaucracy to something that doesn’t work well.

Are they underperforming?

It’s excellent that agile has become mainstream over the last 20+ years. Agile has done wonders for how frequently we can ship software. The norm for delivering stuff has shifted from once or twice per year to bi-weekly or even quicker.

But just look at your phone and read the release messages of apps like LinkedIn, Facebook, YouTube, Spotify, or any other prominent app. They’re mostly ”bugfixes and stability improvements”, rarely new features.

So, **while the release frequency has increased, most of the team's efforts seem to be focused on fixing broken things**, which wasn't the idea behind agile.

If you want to know if teams could do better, just ask this: *"Do you feel you are working at your peak ability, or do you think that if you changed some things, your motivation, quality, and speed could be improved?"*

I have never met a team that says they are even close to their peak ability. Usually, there are some areas that, if improved, the team would be noticeably better in. So, **fix some of those issues instead of hiring more people** for increased capacity.

Benchmark against yourself

To know that you are improving as a team, you need to measure your progress. We are measuring to see progress, not to get an absolute benchmark against other teams (to avoid gaming the numbers).

As I wrote in the chapter "Don't start at the wrong end," there are three areas to monitor when improving teams and organizations: motivation, quality, and speed.

Motivation (engagement) in a team can be measured by happiness index, employee NPS, or similar. The important thing is that you measure this often (weekly or monthly) and that it requires little effort from the participants. Focus on trends (is motivation going up or down) rather than absolute values.

Actual quality is measured in production. We measure what we expose our users to by counting defects found, incidents that occurred, customer service requests, star ratings in the app store, and anything else where our users let us know how they experience our work. Most organizations already have these numbers, but they are rarely fed back to the development teams in a structured manner.

Speed is measured by lead time, i.e., how long users must wait for a new feature from when we first started talking about it until they can experience it. Bug fixes and stability improvements don't count (they are necessary evils that prevent us from adding valuable features to our product). When you ship a new feature, just track backward in time to the point when you decided this needed to become part of your product.

Finally, there is one more thing for every team to measure. When I worked for Skype, we called it **Evidence of Continuous Improvement**. Track the number of improvements the team has completed (or new habits adopted) every month. The size of the number is unimportant; other than that, it should be >0. A high number is not necessarily better, but we do want some evidence that continuous improvement is happening.

Don't forget to celebrate when you have improved!

You can't really call yourself a high-performing team if you aren't measuring. It's the difference between thinking you are improving and knowing you are. It also demonstrates that small improvements add up over time. This is generally hard to see in the heat of the moment, but it is so rewarding as a team to look back and notice a nice trend. Remember to celebrate with a beer!

Self-improvement in the team

In my experience, I have found there are a few areas where the majority of development teams can improve:

1. Engineering practices
2. Team composition
3. Agile practices

Of course, more areas could be on this list; these are the most common and can make a noticeable difference.

Engineering practices

Automate “everything”.

There is no reason to have repetitive manual procedures in a development team that includes programmers. The mantra should be “automate everything”, or at least everything you must do more than twice. This includes automating:

- Setting up your development environment
- Setting up your test environment
- Building your product
- Running unit tests
- Deploying your product
- Running user story tests, integration tests, and API tests
- Upgrading your database schema
- …

When you automate things like this, you save time and ensure consistency in the procedures, regardless of who initiates them.

Many teams have some automation, but surprisingly, deployment and database upgrades are often missing, making it cumbersome to set up new development and test environments.

Manage your test environment.

The goal is to deploy and run the part of the system a developer is **working on locally without having to access a shared database or shared external services.**

Besides automating the building and deployment of your product, a productive development environment often also requires:

- Creating a test and development database with known contents that is small enough to be quickly dropped and restored at any time.
- Replace any external services with test doubles run locally that emulate parts of the behavior of the external system.

When the above is achieved, it's easy to automate all kinds of test cases that require a deployed system and spin up new test environments whenever necessary.

Pair programming and/or mob programming

There has been enough research showing that pair and mob programming are faster than solo programming, produce code with fewer bugs and higher maintainability, and enable cross-training within the team.

If programming were about typing characters on a keyboard, two solo programmers would be more efficient than a programming pair, but that's not the case. Programming is about problem solving. It's a stream of small problems that need to be solved to figure out which code to write to make the software behave the way we want.

Driver + co-driver moving fast – just like pair programming.

Problem solving is best done in pairs or groups. We solve problems faster and better when we have others to discuss them with – “two heads are better than one,” as the saying goes.

Code reviews using pull requests are inferior for ensuring we have solid source code and for sharing knowledge. The reviewer only looks at what has been changed in the code, not what may have been missed, and gets very few clues about the thinking behind the code.

Most pull requests lie around for 1-3 days before being merged. That’s a waste and not even close to continuous integration.

A way to move away from pull requests and ease into pair programming is Pair Review. Before pushing your code, you must sit with a programming buddy and explain it to them. If you find improvements that can be made, you do them then and there. Now you are pair programming a little bit!

Team composition

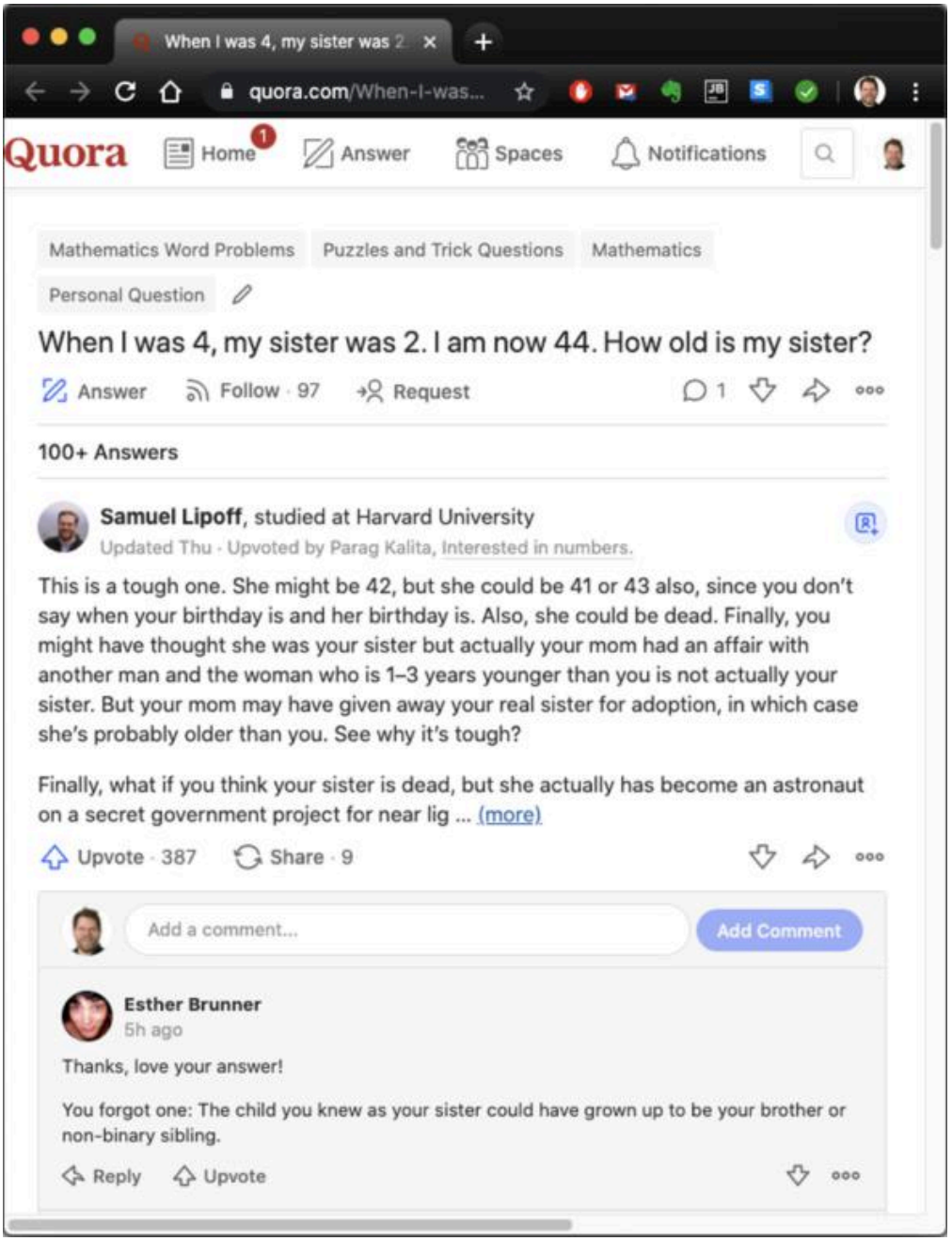
The purpose of having cross-functional teams is that during normal development, you don’t have to reach outside the team to access expertise or someone with a higher mandate. This removes external dependencies and makes the team faster.

There are two more things to consider in terms of cross-functionality, though:

1. The team may not realize they are missing a vital skill set within the team
2. There can still be internal dependencies within the team that hinder their speed

Have testing competence within the team

An often overlooked area of expertise within teams is professional quality assurance and testing. Testers approach a system under test differently from developers. Generally, developers want to prove that the system works, while testers want to confirm that the system isn't broken in any way and measure the quality level so informed decisions about what to do next can be made.



Testers think about everything that could give an unexpected result.

Every team member should be available for testing duty, but it helps a lot if that work is led by someone who is a trained professional and knows what they are doing.

Working with real test specialists has been a truly inspiring experience for me as a developer, especially when you remove the “us and them” attitude.

Ben Simo recently wrote this in a post on LinkedIn:

> *Testing isn’t just about demonstrating that software can work—it’s about discovering how it may not work as desired.*
>
> - *Test to discover what is undocumented.*
> - *Test to discover what is unsafe.*
> - *Test to discover what is unintended.*
> - *Test to discover what is unpredictable.*
> - *Test to discover what is unknown.*
>
> *If you aren’t seeking to discover, you may not be testing.*

Everyone should be a generalizing specialist.

In many teams, you may hear things like, *“We don’t have enough work in this sprint for our front-end developers, so let’s add some stories for them to do.”*

No! You are not hired as a front-end developer on your team; you’re hired as a developer specialized in front-end work. The role is “developer” or, even better, “team member.” Your specialty is front-end development.

The team’s goal in a sprint is to deliver the most important product enhancement (the one at the top of the product backlog) to the end users and the market, and every team member should prioritize that.

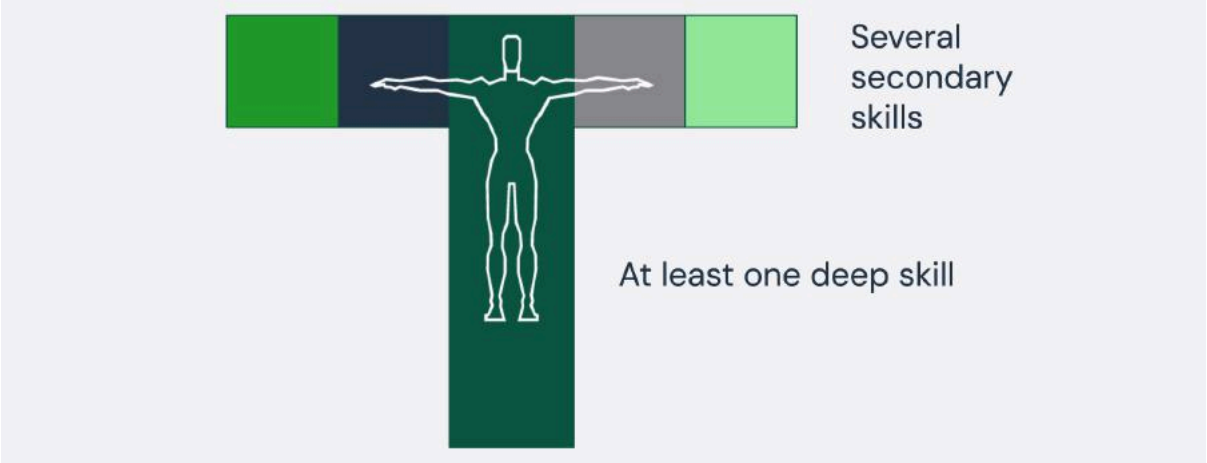
This means sometimes working outside your primary expertise to get things out the door. It’s about **optimizing the team effort, not maximizing your personal productivity**.

So, if there isn’t enough front-end work to do, you can help out with other team responsibilities like:

- Back-end development
- Testing
- Discovery work
- Automating things so that others become more productive
- …or anything that shortens the lead time for the current product enhancement or makes the team as a whole better

So far, I have used front-end developers as an example, but the same goes for any type of specialist within the team, such as back-end developers, testers, product owners/managers, Ux specialists, etc.

When planning a sprint, it's a common mistake to implicitly assign everyone some of the work so they can carry on doing what they are best at and prefer to do. This is a suboptimization that slows the team as a whole.



The more senior you are, the wider your skillset should be.

I'm not advocating for everyone in a team having to be an expert at everything. Not at all! Having specialists in a team is often a blessing. However, everyone should have a few secondary skills to help out where they are most needed to optimize the productivity of the team as a whole.

Agile practices

Underestimating the power of Team Coaches

Being a ScrumMaster or Team Coach is a real profession, and it requires a very different skill set from, for example, being a programmer. Many teams underestimate this and view Team Coaching as mainly an administrative task.

However, the primary responsibilities for a Team Coach are:

1. The well-being of the team
2. The team's ability to deliver valuable stuff fast and with high quality
3. The continuous improvement of the two above

That list includes very little about booking meetings, managing JIRA, or facilitating ceremonies like daily stand-ups, retrospectives, and demos. Those activities are only tools to help teams reach their potential.

The Team Coach should aim to make themselves redundant so that the team members manage every aspect of being an agile team. That's when you have a genuinely self-organized team.

Unfortunately, in most organizations I have encountered, ScrumMasters and Team Coaches consider this role part-time work and don't make it their first priority. When pressed for time, they choose to contribute to the team by writing code, performing tests, or whatever they see as their primary job, rather than making it easier for their teammates to perform.

Again, this is a suboptimization. A Team Coach is a lever for the capabilities of the team. If they can, in a 5-7 person team, increase the team's productivity by just 20%, their efforts will be worth the time invested. From my own experience, and when talking to other experienced ScrumMasters and Team Coaches, we all agree that you **can double the performance of an average team within a year** if you put in the effort. That's much better than hiring twice as many people and expecting capacity to scale linearly.

Low-value retrospectives

Regular retrospectives and continuous improvement are, by far, the two most important Lean/Agile practices. There is always something that could improve our work situation since we never get everything right the first time, and things around us keep changing.

Although most teams have retrospectives regularly, their value diminishes over time in many cases as enthusiasm fades and the easy problems are already addressed. This leads to shorter retros (maybe just 30 minutes), and the improvements decided on are the same things that have been talked about before. **This is not improvement work. It's just going through the motions.**

To learn if a team is stuck in this rot, here is an experiment you can try: After the retrospective, ask everyone in the team to write down on a sticky note the answer to; "What sucks the most in your professional life right now?"

Let everyone show their notes and then discuss whether the issue was brought up on today's retrospective and, if not, why not. Maybe you need to improve the retros themselves.

As a rule of thumb, you should always work on the team's worst problem, even if it seems very difficult to overcome. Your work situation can only be improved if someone is working on it. In the next chapter, "Continuous improvement on an organizational level," I will cover how to support teams in doing that.

Practicing and deliberate learning

Finally, continuous improvement is not just about retrospectives. While that's one method of improving a team, another is deliberate learning.

Any top-level sports team or serious rock band spends significant time practicing and deliberately learning new things to become better at their profession. There is no reason why business teams (like a development team) should not do the same.

Practice makes perfect. Development teams need practice too.

This is about having a learning curve and staying ahead before the team runs into problems (that may be caught after the fact in a retrospective). The best teams I know have regular learning labs, code katas, knowledge sharing, training sessions, etc. It helps the team to keep up with technology and mature as individuals in their profession.

When was the last time your team deliberately practiced just for learning?

Conclusion

Before scaling up with new hires, optimize the teams you already have. If you measure, improve, and empower teams to perform at their best, there is a good chance you can double their capacity without hiring a single person. Isn't that something?

Continuous improvement on an organizational level

Over time, the real competitive edge is an organization's ability to improve continuously. It's like paddling upstream on a river. If you stop paddling (making small incremental improvements), you'll soon start drifting backwards towards the waterfall.

Agile has shown us the importance of regular retrospectives and continuous improvement at the team level, but improvements need to happen throughout the entire organization. Just like there is an organized way to improve in an agile team, there should be an organized way to improve at every level.

Let's take a look at three steps to expand the improvement culture beyond the grassroots team level:

1. Be strategic about your capabilities
2. Install escalation paths
3. The improvement mindset is for everyone

Following these steps will help you build the (organizational) system that yields the desired business results.

Be strategic about your capabilities.

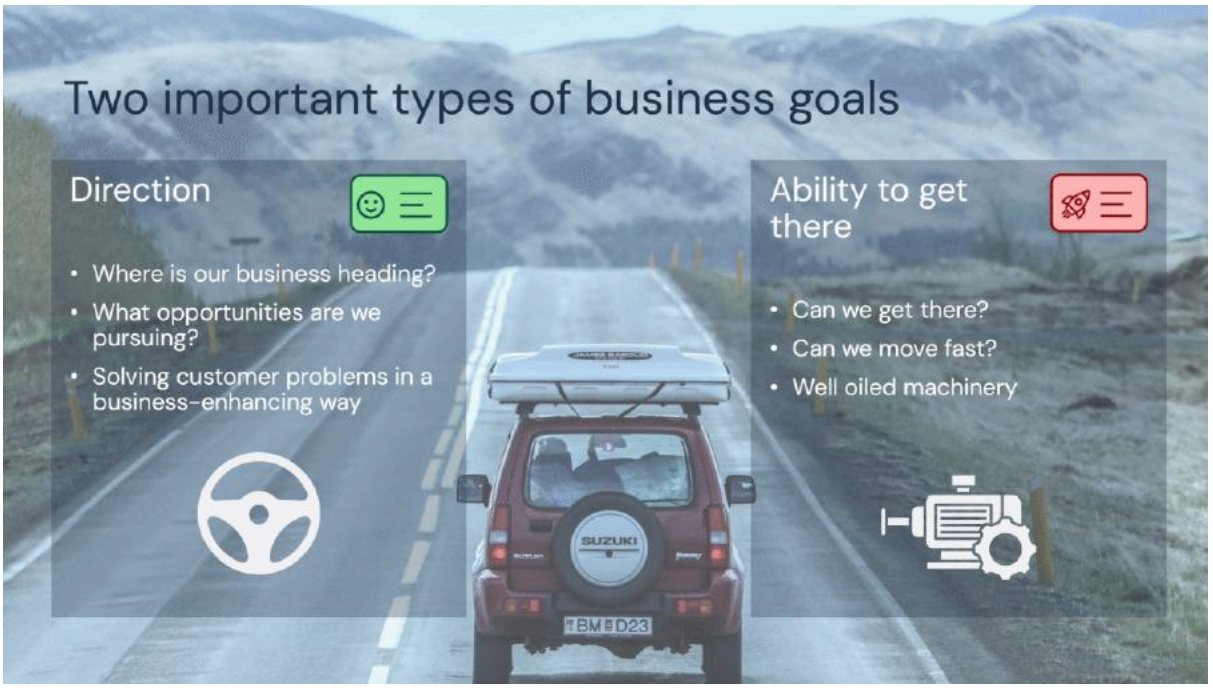
The steering wheel & engine

Strategic business goals set a direction for a company. We describe where you want to be in the future as a business. To get there, we carve out a strategy that guides you along the way.

Let's compare this to travelling from Stockholm to Barcelona. Barcelona is the goal, but we won't get there in one go, so we roughly plot a route (a strategy) through several countries and expect to reach the goal in five days.

What would stop us from reaching our goal in time? If we drive a well-serviced car, there will be no problem getting there. But if we drive a vehicle that hasn't been serviced in the past five years and hasn't been properly cared for, we might not get far before we run into trouble and run out of time.

As a company, we can set the best strategic goals and have the best strategy to get there (turn the steering wheel in the right direction), but if we lack the capability (the well-serviced engine) to get there, that doesn't matter.



When setting strategic goals for an organization, they typically fall into two categories: business goals and capability goals, which ensure you can achieve the business you desire. The strategic insight is this: A capable company can competitively reach almost any business goal.

So, what are these capability goals? It depends on what your business goals are. What capability does your company need to have to reach your strategic business goals in time to be successful? Often, the types of goals have to do with.

- Lead time (time to value)
- Quality
- Agility and Flexibility (ability to deal with variation)
- Competence and skills (outsmart the competition)
- …

Set the standard

Having self-organized teams that execute all the essential work in a company doesn't mean they do whatever they want. There has to be a direction (goals and strategy) that tells us where we're heading and constraints (rules) that describe what's expected behavior.

Compare to a soccer team: They know where the goal is, who the opponents are, and what is meant by winning. They also see the playing field and understand the game's rules. Knowing all that, they can play a fair game in a self-organized way, where the team handles defense and attack on their own in real time.

For agile teams in a company, it means we have to define a bar that they must stay above. Standardization is a good way to do this. Define what is expected from any team in the organization. They are allowed (and even encouraged) to exceed these expectations, but they should at least do some basic things. Here are a few examples:

- Every team should have a mindset of continuous improvement and have at least one improvement in flight at any time. Improvement isn't optional, it's expected.
- Every team should benchmark against itself using the metrics described in my previous article "Part 4: Don't scale with underperforming teams".
- Every team should have regular retrospectives or some other well-defined way of improving their motivation, quality, and speed.
- Every team should know and be able to name their stakeholders (you'll be surprised…)
- Every individual should have a learning plan to deepen and broaden their skill set.
- Every team should have 1-3 objectives in focus and prioritize their work accordingly.
- The company's objectives should be supported by the team's objectives.
- …

Rules like this not only set the bar but also provide transparency, ensuring that people have the information and tools they need to make wise decisions at every level.

Coaching from the sidelines

I often find that organizations talk about having things like this, but they don't enforce them as company rules. In my opinion, it's a mistake to be passive about this.

Although managers should remove themselves from the playing field to the sidelines (let the teams self-organize), they should not move up to the spectator seats. We want managers to coach their teams actively, and to do that, you have to be close without being in the way.

Operational excellence

Driving business strategy and increasing the organization's capabilities can be spearheaded by the C-level executive team, but some companies establish a dedicated team focused on operational excellence. That would be similar to what Jeff Sutherland calls an Executive Action Team in his Scrum@Scale framework.

The operational excellence team is the top level of company improvement and capability building, serving as the final escalation point for issues that cannot be addressed by teams at lower levels (see below about escalation paths).

Sometimes, a group like this is called a transformation team and is created when a company decides to adopt new ways of working in a focused approach, such as transitioning to Agile, SAFe, the Product Operating Model, Scrum@Scale, or a similar approach. Since continuous improvement never ends, it's always my advice to make this constellation permanent and continue serving the organization even after the transformed state has become the new normal.

Most members in an operational excellence team also have other jobs within the organization. It's beneficial to have change agents from multiple corners and levels within the company. Not the least a strong representative from the C-level team.

The operational excellence team sets capability goals and objectives at the company level, as well as funds, launches, and follows up on improvement

initiatives executed in other parts of the organization. They serve more like product owners or stakeholders than an implementation team.

Don’t become a victim of your own success.

Startups and scaleups are usually less strategic than more mature companies. They are focused on generating cash flow and growing their business in any way possible.

This is understandable, but if you don’t think about “servicing your engine” and things take off, you might end up with problems caused by demand.

Thanks to your success, you have more users and more feature requests, and you need more capacity to keep up. This may lead to:

- Poor system performance. Your system was designed with dated specifications and can’t handle exponential growth.
- Rapidly rising technical debt. It’s too complicated to add new features in a proper way (not just patching them on), and re-architecting is not on the table.
- Productivity drops despite hiring more people. When you become four times bigger in a short time, only 25% of the staff have been there from the beginning and understand the business and the system internals.
- Onboarding takes an unnecessarily long time, as there is little documentation, the old team members are busy, and the system has become a patchwork of solutions.
- … and so on

With some foresight, this can be avoided. If your business projections (the ones you use when attracting investors) indicate a 10x increase in usage, global presence, and a roadmap with some fantastic features that need to be built you should make sure you have a well serviced and well tuned engine (organization, architecture, skilled staff and processes) to get there.

Set capability goals that align with your business ambitions, so you don't become a victim of your own success.

Install escalation paths

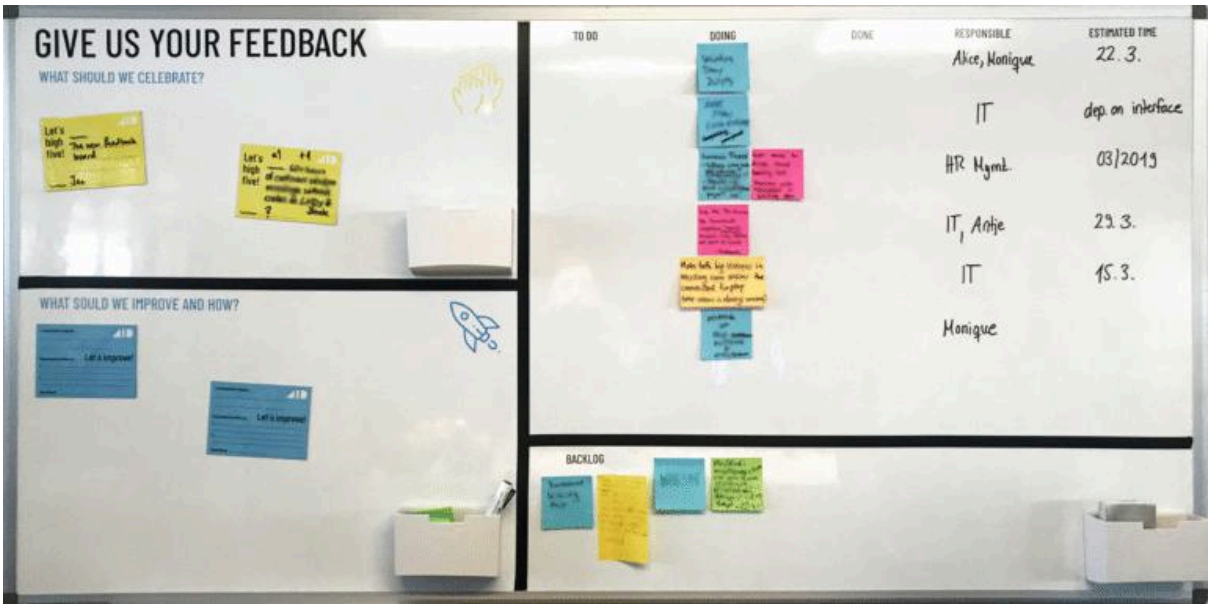
We want to make being a fast organization a permanent state. This means that the leadership has to build an environment where people can perform and have a system for identifying things that suck and fixing them when they can't be addressed on the team level.

Here is where escalation paths come in. If a team's worst problem is something they cannot resolve on their own because it requires managerial powers, affects a larger part of the organization, or has its cause higher up in the decision-making hierarchy, there must be a way for the team to escalate this issue and get help.

In many organizations, the "next level up" is typically the closest line manager, but this may vary. Some companies have a team of discipline managers (for programmers, testers, and product managers, respectively) overseeing a cross-functional team.

It's the duty of the closest management level to help the team. **Any manager who doesn't help their team with its worst problem is not doing their job.** However, it's possible that this manager also lacks the power and influence to resolve the issue, so there must be a way for them to escalate one more level. And so on…

The responsibility ends with the operational excellence team. As mentioned above, they serve as the final point of escalation for issues that cannot be addressed at lower levels.



This company improvement board was placed in the coffee lounge, where everybody could see what the leadership was working on.

Transparency is key. The available escalation path should be visible and transparently show what issues each level is currently working on to resolve.

A colleague of mine implemented the following with a client a few years ago: Each manager for a group of teams had a small whiteboard outside their office, with space for up to four sticky notes. On each sticky note, a problem that the manager was helping a team to resolve was described. Any team could put a sticky note on the board, but the total number of issues could never exceed four. If there was no more room, the teams could negotiate with each other and replace one note, relieving the manager from that work. When an issue was resolved, it was up to the team to remove the item from the board, not the manager.

This experiment turned out to be successful. Due to the WIP limit of four, managers were not overloaded with issues to fix for the teams, and the teams had to be careful not to use up space with unimportant items. Between the managers, there was a bit of prestige to keep their board as empty as possible.

The improvement mindset is for everyone.

When was the last time your executive team had a retrospective? Does it happen every two weeks, like in most development teams?

The habit of regular retrospectives and continuous improvement should apply to all teams in an organization, not just “worker teams” such as development, operations, and support teams.

Actually, **it's not uncommon for some of the worst teams in an organization to be leadership teams** with low psychological safety, poorer-than-average collaboration, and only sporadic deliberate learning and self-improvement. Learn from the agile teams in your development organization. They can show you how to do it!

Don't just focus on improving others – take a look in the mirror as well.

Summary

Continuous improvement concerns the whole organization at every level, not just contributing teams.

Like in agile teams, company-wide self-improvement needs to be structured and deliberate. I have suggested some ways of achieving that.

When setting strategic business goals (turning the steering wheel), ensure the organization has the matching capabilities to get there (a well-tuned engine and a mindset of constantly improving it).

About the author

Jan Grape is an organizational improvement guide at Snowdop AB who has been in the software product development industry for over 40 years.

He earned his first money as a developer in his teens, creating a CAD program used by schools in Sweden.

Throughout his long career, he has worked as a developer, usability engineer (Ux), architect, team lead, teacher, manager, CEO, agile coach, organizational coach, and product coach.

In his profession as a consultant, Jan is dedicated to helping organizations improve themselves, creating awesome workplaces where people thrive and produce great products.

https://www.linkedin.com/in/jgrape

SNOWDROP

snowdrop.se