Leading Snowflakes

The Engineering Manager Handbook

Practical tools and techniques for programmers who want to lead

BY OREN ELLENBOGEN

Leading Snowflakes

The Engineering Manager Handbook

Practical tools and techniques for programmers who want to lead

BY OREN ELLENBOGEN

Leading Snowflakes

The Engineering Manager Handbook

Oren Ellenbogen

To Shahaf, Evelyn and Joshua
for your love and support

Thank you for supporting this endeavor. It is my hope that my lesson learned, transitioning from an engineer into a manager position, can help you to save both time and pain in your own journey as a leader.

This eBook is not open source. I am a big advocate of fair use and chose to avoid any form of Digital Rights Management in order to ensure you can enjoy this lesson in whichever way works best for you. If you received a copy without paying, please consider purchasing it.

http://leadingsnowflakes.com

If you have any feedback or questions, please reach out to me. I'd love to hear from you. You can email me at oren.ellenbogen@gmail.com or contact me via Twitter @orenellenbogen.

Acknowledgements

First and foremost, I'd like to thank Talya Stern for asking the tough questions, helping out with copy editing, spell checking and grammar. I couldn't have done it without you.

I would like to also thank the early beta readers of this book: Bill Hodgeman, Eyal Efroni, Ohad Laufer, Tomas Lundborg, Victor Trakhtenberg, Ilya Agron, Effie Arditi, Shani Raba, Amit Yedidia and Maxim Novak, for giving insightful and helpful ideas and corrections.

Lastly, I'd like to thank Nathan Barry, Justin Jackson, Adii Pienaar and Paul Jarvis for not only inspiring me to write this book, but also for teaching me a new approach for self-publishing.

Words can't express my gratitude for what they've contributed to this book.

Book design by Mark D'Antoni.

Contents

Mission Statement

The goal of this book is to help Engineering Managers or aspiring managers become more effective at leading software engineers by improving their ability to retrospect, communicate, collaborate, delegate and inspire other people.

Introduction

Alone.

If I had to choose a single word to describe how I felt in my transition from an engineer into a manager, that would be it. As much as I was excited, I was also terrified.

We were never taught the dark art of building a team or leading people. As if this isn't enough, these unique individuals (hence "snowflakes") in our team tend to be incredibly smart, analytical, opinionated, self-driven, and ambitious. Engineers.

What is it then, which could aid me in finding my own management style? What can I do to gain that confidence I need, to start leading, rather than reacting?

One of the things I appreciate most in our profession as software engineers is being able to break complexity into smaller, almost tangible parts, where figuring out these patterns of simplicity can not only produce beautiful solutions, but also introduce us to the building blocks of beautiful software. Writing this book was my own journey to present tools and techniques I believe can discover these building blocks of beautiful leadership.

Now, our job is to ask the right questions, to encourage people to think, to challenge, to inspire.

While we may always feel that we are walking in this path alone, I want to challenge you to look deeper into the way you lead others. I want to challenge you to share the reasons you lead people in a certain way, so those patterns of simplicity you find along the way can become your own secret sauce to leadership.

I'm cheering for you for taking the time to invest in yourself and do what you believe is best for your team. I know that you're busy, that you hardly have the

time to breathe. It is people like you who encouraged me to write this book. I know because I've been there. I know, because I can feel what you are feeling now.

I was never a consultant or a theorist. My pains came from facing the same problems you're facing today. My desire for patterns and practices came from mentoring other Engineering Managers along the years.

It is my hope that my observation and techniques could enrich your managerial arsenal, helping you to find your own path to become a great Engineering Manager and a great leader.

How this book is built?

In writing this book, I wanted to make sure you will learn practical techniques you could start using immediately. This is why I've made sure that each lesson is concise, self-contained and actionable.

You'll find each chapter to start with the motivation behind the tools and techniques you're about to get to know, followed by a list of tasks you can use to track your progress.

Let your curiosity lead you, as you can navigate between lessons the way you find most interesting and relevant.

Finally, I've created an online application you could use to track your progress across all lessons. To do that, simply sign-in to http://leadingsnowflakes.com.

LESSON 1

Switch Between "Manager" and "Maker" Modes

Switch between “Manager” and “Maker” modes

THIS LESSON IS ABOUT:

1. Understanding how interruptions have a different effect on Makers and Managers.
2. How to use your calendar and small gestures to create the quiet time needed to “get into the zone”.
3. Figuring out which types of tasks you should own on your “Maker mode”.
4. Tactics for finding the right balance between Maker and Manager modes.

Time investment: 25 minutes.

Motivation:

We all started our professional journey as a “Maker” – a designer, software engineer, product manager, tester or operations engineer. Our ability to make things is what brought us so far.

We all love the feeling of getting things done.

This is why when we go from a “Maker” into a “Manager” role, we so often fall back to our comfort zone. We're neglecting managerial responsibilities working to complete yet another task instead. It's the power of old habits. Also, it's much more fun (and immediate!) when getting the work done is completely in our own hands.

But let's face it – no one is being promoted to a managerial position to increase their own productivity. **Our job as managers is to amplify our teammates**. No longer will we be measured by our own ability to complete tasks. We already

proved our capabilities there. We will be measured by a new unit of execution – team execution.

One way to avoid stepping back to our old habits is to completely commit to our new role and delegate all of the "Maker" tasks to our teammates. While this move can feel "right", as it forces us to focus on scaling our teammates, this decision can be fatal in the long-term. By completely disconnecting ourselves from the details, we are risking our ability to understand implementation complexity. This may lead to losing our edge as someone who's able to help with prioritization and estimation, both internally and externally. Our teammates and peers will soon "smell" our incompetence, and once they will feel we are no longer contributing to technical debates or prioritization, they will stop consulting with us, even if unintentionally at first. **The last thing any manager would like is to own control by forcing it.**

And so, as Engineering Managers, one of our hardest challenges is to carefully balance these two modes – Maker and Manager.

To find that "sweet spot", we need to understand how interruptions (e.g. meetings, questions) affect Makers differently than Managers. We need to understand what kind of tasks we can and should take on ourselves as Makers, without completely losing our attention to our teammates or our management focus. Lastly, we need to start utilizing our calendar better and have a few signals to communicate our transition between those modes during the day or week. These subtle gestures will allow our teammate to give us the quiet time needed to get things done.

Finding that balance between these two modes takes time, practice and a lot of adjustments. The good news though, is that **there are actionable steps we can take to practice and optimize our ongoing transitions between these modes**. We're going to cover these steps now.

How interruptions affect Makers and Managers

Before we dive into the reasons why interruptions behave completely different for Makers and Managers, we need to take a step back and figure out the expectations from each role.

As Makers, we are measured by our ability to **complete** tasks. There are many moving parts and the required context, for most tasks, is pretty large: balancing client requirements with project constraints, performance and scalability considerations and the list goes on and on. This context and the nature of some of these tasks ("part art part science") are the reason why Makers need a lot of time to "get into the zone".

As Managers, we are measured by our ability to **scale** our team and company. That usually means improving one or more of the company's KPIs, clearing obstacles, training our teammates, communicating progress, measuring our team's velocity, experimenting with different processes to increase the team's velocity etc.

These different expectations lead to very different modes of work.

As Makers, we prefer to have as many continuous hours as possible, focused around getting things done. No meetings, no short status questions, no 30 minutes of "check out my new idea". Complete silence.

As Managers, we prefer to have many small wins by helping our teammates. We enjoy running around, asking "what's up?" and "how can I help?" We love to collect small wins by talking with people. Interruptions are an integral part of our job.

More emails than GitHub issues? Welcome to the maker/manager transition...

— Tom Moor

Enabling “Maker focus”

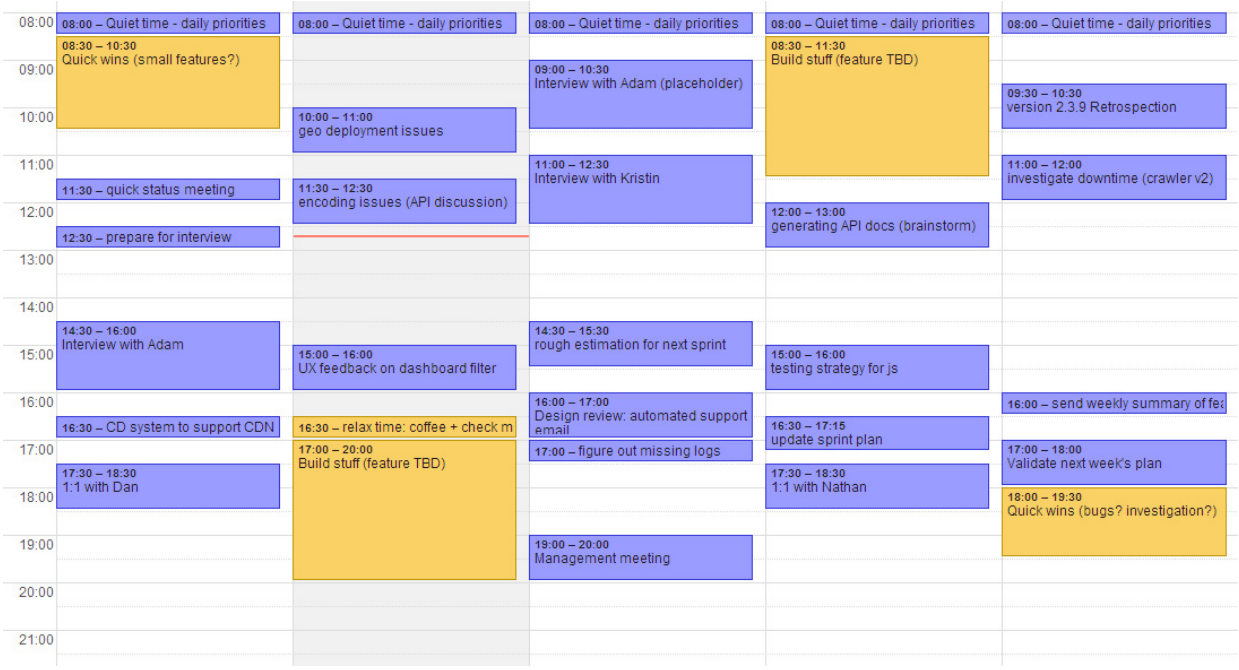
Our time is the most important asset we have, yet we are usually passive when it comes to planning our day. I've done it for many years, working a full day as a Manager, and then continuing late into the night as a Maker. “I'll create a plan next week, after this disaster will be done!”

Using our time poorly is the best way to guarantee burnout.

If we won't block the time needed for the Maker mode, someone else will. It could be our boss, one of our peers or some random (not really) urgent matter that would pop up.

“Role-based” calendar

A small trick I found useful is to visualize my calendar using different colors. I've created two calendars – One for the Manager (blue) and one for the Maker (yellow). For the Maker, I've blocked in advance specific hours during the week so no one else could set a meeting for me at that time. I like to use early mornings or towards the end of the day to switch to my Maker mode. Here is an example:



Sometimes, we may need to change our plans, but at least we're starting the week with a game-plan we'd like to see coming through. Using this kind of view, we can easily see that we have blocked enough time for getting things done. We are in control.

A note about "Quiet time – daily priorities"

My favorite practice was sitting at home with some hot coffee or tea when planning my day. Having a quiet environment made it a much easier and shorter challenge. In most days, I was ready to go by 08:15.

Using "do not disturb" gestures when switching modes

We don't want to completely disappear to our teammates, but it's imperative to have an option to tune-out. Running around in the hallway, yelling "I'm about to change into my Maker mode, heads up!" is not a great option (although it might be funny to try out).

We need more subtle ways to signal our team that "I'm trying to focus here, is it really important? Can you send me an email instead? Can you handle it yourself ?"

Small gestures can be a great way to subtly indicate our current mode. Here are a few ideas:

> **Headphones** – Wear your headphones every time you want to signal you're in Maker mode. It's a great way to disconnect from the outside world and fully invest yourself in the task.
>
> "**Focus room**" – Find a quiet room at the office (or at a nearby coffee shop) and sit there with your laptop for a few hours. Not only is it a very clear signal for your teammates, it's also a great way to break your frantic working routine.
>
> **Start the day working at home** – Sometimes it's easier to continue working as a Maker once I'm done planning my day. It still leaves me with enough hours to be available at the office for meetings or helping my teammates with daily challenges.

Create your own ways – I'm sure you're much more creative than I am, so I challenge you to come up with interesting ways to signal it. Wear a funky hat? Hang a red "ON AIR" neon light outside the room? Keep it light and playful, you don't want to scare off your teammates, or do you?

On your next team meeting, share your experiment with balancing your Manager/Maker modes. Explain why you believe it's crucial for you to remain hands-on, and that you will need their help and understanding as part of this journey. If you feel it's too soon to share it with everyone, start by sharing it on your 1:1s and see how they feel about it.

Figuring out which tasks we should own

While we don't have to be the most technical individual on the team, we have to be someone our teammates can go to in order to get help with prioritization and risk analysis. The only way we can keep our edge is by continuously practicing our Maker skills, even if it's on a smaller scale.

Here are a few guidelines I've used along the years to help me pick my tasks:

Keep them small (<4 hours) – pick tasks that are fairly simple and can be completed in less than 4 hours. It may involve investigating a bug or a performance issue. It may include creating a few mockups or suggesting a new UX for some feature. Whatever your job is, pick something you're confident will be small enough to fit. You should still have capacity for managerial responsibilities during that day.

Whatever your backlog system is, make it easy to pick small tasks you can take. If you've got 3 hours to be productive, it would feel counter-productive to use this time to think of tasks, instead of actually making them happen.

Not on the critical path – the last thing you want to do is to become a technical bottleneck. Remember that your goal is no longer to

complete as many tasks as possible nor it is to complete the most important ones. We should prefer to be involved during the design phase and delegate the implementation work to our teammates. At start, it would be very difficult to handoff such a huge risk to someone else. Be there to support and mentor them in this transition, but let them own it.

Boost our teammates' velocity – if you crave for bigger challenges, I would pick tasks that will benefit the velocity of the team. It can be experimenting with a new infrastructure, reducing Technical Debt or maybe creating a tool to cut down needless work. I find this approach both fun and valuable – I feel productive both as a Maker and as a Manager, plus I win my teammates' appreciation.

Explore a new technology or process – can you think of ways to replace an existing capability by using something completely new and exciting? Try to produce a working demo and present it to the team. It's okay for it to be a throw-away; sometimes, the process of learning a new way to accomplish things can help you in brain-storming and challenging your teammates.

Boost our company's ability to attract and retain talent – Can you create a website that would show how awesome your team or company is? Can you write a small technical challenge and publish it online so people around the world could try to solve it? Can you publish a small open-source project which utilizes your own company's API to do something completely crazy? Can you publish a guest post about an interesting dilemma your team was facing? What about preparing a 30 minute lecture to share some lessons learned?

Boosting your company's appeal for new hires can do wonders to your ability to attract talent inexpensively. That's a huge win!

Iterate until you find the right balance

The first lesson I've learned was to change the number of hours I've spent each week as a Maker. Once a month or so, decide how many hours you want to invest

every week for the following month. You can start with 15-20% Maker time, while having the rest of the schedule open for managerial tasks. If your team is small (2-3 people), you could probably invest up to 50-60% of the time as a Maker. Of course, **it also varies according to how strong and experienced your teammates are**.

As your team grows, you may feel the urge to drop those hours completely. Don't. Instead, keep a minimal 4-5 hours on your calendar each week.

Use your 1:1 meetings to get feedback

The best way to learn is by asking for feedback. Share what you've learned so far on your 1:1's with your boss and your teammates. Ask for their opinion on your availability and attention.

You keep cancelling your Maker meetings?

When nothing works, I always go back to a single practice which is starting the day **working from home** for a few hours once or twice a week. Feeling that small accomplishment of getting things done keeps me on track and reminds me the importance (and joy!) of staying technical. Also, I would recommend **finding some more** Hackathons **you could participate at** – There are plenty of Hackathons you can join, not even necessarily in your company. Use that time to demonstrate your capabilities and improve your skills, guilt-free!

Task List:

So, what should you do next?

- ☐ Re-organize your calendar: block enough time for Maker mode – I suggest no less than 2 hours at least twice a week.
- ☐ Create a clear color distinction in your calendar, for "Manager" and "Maker" time.
- ☐ Make sure you've got at least 5 "small wins" in your backlog that you can start implementing tomorrow.
- ☐ Pick a gesture you can use to switch modes during the day.
- ☐ Discuss it with your teammates on your next team meeting: ask the team for advice and feedback along the way.
- ☐ Read Rands' post "Technicality"
  (Time Investment: 12 minutes.)
- ☐ Read Paul Graham's post "Maker's Schedule, Manager's Schedule"
  (Time Investment: 10 minutes.)
- ☐ Share these posts with other managers in your organization. Have a discussion over lunch.

Track your progress online!

I have completed this session

LESSON 2

"Code Review" Your Management Decisions

“Code Review” Your Management Decisions

THIS LESSON IS ABOUT:

1. Creating a system to easily peer review our decisions as a manager.
2. Actively identifying recurring problems in our team.
3. Self-retrospecting old decisions we've made, that we'd want to handle differently from now on.

Time investment: 15 minutes.

Motivation:

When I was first promoted to an Engineering Manager position, after being a technical lead for many years, I soon started to feel how badly I miss the **short feedback loops** I used to have. Unlike managers, programmers have a variety of methodologies and tools such as Pair Programming, Code Reviews and Design Patterns, just to name a few. These processes used to help me to become more effective and improve my engineering skills by **using other people's experience and knowledge**. I missed that feeling.

As managers, especially as first time managers, we tend to feel completely alone.

We don't want to admit that we don't have a clue to our boss or our teammates. We're afraid to show how frightening it is to be responsible for someone else's success. We're constantly worried about balancing development risks and business needs.

“Fake it ‘till you make it”, I remember I kept saying to myself. I felt like a fraud.

I knew that “People quit their boss, not their job”, so I decided to break my old habits of hiding my dilemmas and start looking for ways to openly seek for feedback and improve my management skills. I wanted to be great at my new position, to inspire my teammates, to challenge them, to help them shine.

The same passion that drove me in my days as a programmer, was guiding me now.

I would like to show how some of these peer review techniques can be applied to our everyday activities as a manager. By proactively reviewing our decisions as a manager, we should feel more in control of our progress and learn more rapidly from our surroundings. Once there's such a system in-place to capture our decisions (or lack of them), gathering feedback should become much easier.

Receiving constant feedback is the fastest way to learn.

Our boss and colleagues are huge assets to learn from, and – from my experience – are usually heavily underutilized. We are going to change that today.

The impact of a single decision:

Nathan enters your door asking for a few minutes of your time.

He tells you that he needs to refactor some messy code he spotted, while working on his feature. He wasn't sure if he should refactor the code or let Kai, the one who originally wrote it, fix it on his own.

“What should I do?” Nathan asks.

“Hmm... well, Kai is busy with another feature. Fix it and let's move on, we've got a tight deadline to meet”, you reply.

The hidden alternatives

“There is no one right way, there are only tradeoffs.”

There are many ways to achieve the same goal. For example, we could come up with a few more options to help Nathan with his dilemma:

- Ask Nathan to refactor the code and conduct a Code Review with Kai.
- Ask Nathan to do some Pair-Programming with Kai on this change, allowing Kai to learn and improve his skills.
- Let Kai fix it and ask Nathan to work with some dummy data until the code is ready. It's been a long time since Kai wrote this code originally, so it should be good practice for him to refactor his own code.
- Leave the code as it is. Maybe you can deal with it later on as a Technical Debt.

I'm sure there are even more options.

We can imagine the different "realities" each one of these decisions will lead to. Experienced managers would be able to offer a few more options, explain how the other side might see it, or challenge us to put more responsibility on our teammates.

Captain's Log:

For years, I've perfected my own system. I call it "The Decision & Feedback Loop Tracking System"™, also known as TDFLTS:

captain's log:

When	Who	The dilemma	The decision made	Retrospection	Did I share it with someone?

Just kidding. It's just a simple spreadsheet.

Capture our decisions now to discover alternatives later

Writing down our decisions shouldn't take more than 5 to 10 minutes. The idea is to include just enough details to re-ignite our memory on the situation, and provide context to ask someone else for their opinions.

Let's take the example from above, and fill it within the template:

- **When** – *[some date, for example:]* 27th January 2013.
- **Who** – Nathan jumped over to my room.
- **The dilemma** – Nathan told me he noticed some poorly-written code, required for the feature he's working on. He wasn't sure if he should refactor the code or let Kai (who originally wrote it) fix it on his own.
- **The decision** – I've asked Nathan to fix it as Kai is too busy with another feature.
- **Retrospection** – *[leave it empty, we'll get to it later]*
- **Did I share it with someone**? No *[for now]*

Make it a habit

Nothing motivates us more than peer pressure. Here are some tactics to motivate writing things down:

- **1:1 with the boss** — starting from your next 1:1 with your boss, pick 2 dilemmas from your list and conduct a "code review" on your decisions. I recommend that you'll start with describing the dilemma (not the decision!) and let your boss say how she would handle it. Only then, share the decision you've made at the time, and see how she reacts to it. Have an honest discussion, talk about the tradeoffs as you see them.
- **Find another Engineering Manager in the organization you can trust** — ask him to take this journey with you. Set a 30-60 minutes bi-weekly meeting, where each one of you takes 2 or 3 dilemmas, and try to get him to share feedback and thoughts. Just like with your boss, stick to sharing the context and the listen carefully regarding how he would approach it. Note: don't share private discussions, your people should trust you to keep their privacy.

- **1:1 with your teammates** — You can take 1-2 dilemmas raised by the individual you're sitting with, and talk about it. Use this time together to see if you could have handled it better from your side as well. Ask for feedback and guidance just as you offer your own for your teammates. Keep in mind though – some of your teammates don't have the exact context you've got in mind or the required maturity level. That being said, don't underestimate them. They'll surprise you.
- **10 minutes daily ritual** — You don't have to write a new dilemma every day. You can take that time instead to write down a self-retrospection on one of your earlier decisions. It might be best to do it while drinking coffee early in the morning, before you're going to lunch or around the end of the day.
- **1 hour monthly ritual** — spend an hour to review a decision you've made 1-3 months ago. Set a meeting in your calendar to an hour when no one is usually around and you've got no distractions. Consider what decision you would have made, if that same situation happened today. Write it down.

Success is the sum of small efforts, repeated day in and day out

— Robert Collier

Retrospect

Here are a few questions we could use while going over our dilemmas:

The dilemma —

- How many technical dilemmas were raised?
    - Are there any recurring issues that might point out a Technical Debt in our application?

    - Does it feel like the Technical Debt is under control?
- Can we spot some personal issues between our teammates?
    - Can we think of ways to create trust between them?
    - Does it also impact work relations with other teams?
- Is there a specific individual in the team who raises most of the points?
    - Is it because they are lacking some context (team/company goals)?
    - Lacking technical knowledge?
    - They care more about "building it right" (technical lead material)?
    - They care more about their teammates (manager material)?
- Can we spot repeating issues with other teams?
    - Can we solve it with our peer?
    - Is there a need to involve higher level manager in it?

The decision made —

- How many of the decisions were made by us?
    - Did the team really need us to solve this dilemma?
    - Did we ask our teammate to suggest a few options first?
    - Is there someone else who could provide these answers (e.g. the Technical Lead)?
- Would we do something differently, now that some time elapsed?
    - Maybe we should have asked our boss to move the deadline in 2 days, so our teammates could get more value from the given task?
    - Maybe we should let them make the decision and take accountability for it?

Did I share it with someone?

To keep actively seeking for feedback, we should start by sharing at least 50% of the dilemmas we write down.

Task List:

So, what should you do next?

- ☐ Make a copy of the Captain's Log template (examples included on the 'Examples' tab).
- ☐ Set a daily reminder in your calendar for 10 minutes of writing things down.
- ☐ Set a monthly reminder in your calendar for 1 hour of self-retrospection.
- ☐ Find another Engineering Manager in the company to be your partner practicing this approach.
- ☐ Set a bi-weekly 30 minute reminder in your calendar for a peer review with your partner.
- ☐ Share 2 dilemmas with your boss on your next 1:1 session.
- ☐ Share at least 50% of your dilemmas in the first 3 months.

Track your progress online!

I have completed this session

LESSON 3

Confront and challenge your teammates

Confront and challenge your teammates

THIS LESSON IS ABOUT:

1. Breaking emotions from behaviors to push our teammates out of their comfort zone.
2. The "anti-asshole" checklist – a systematic way to care more for our teammates than our own image.
3. 4 common pitfalls to avoid.

Time investment: 20 minutes.

Motivation:

Moving from an engineer into a manager position can feel horrible at times. Not long ago, we were friends and peers of these amazing people in our team and now we're the one "calling the shots" and trying to make this team work well together. Trying to confront my teammates and challenge them to push themselves even further was one of the most difficult parts of my job. It's a huge emotional hurdle.

Trying to remain their friend, I was doing everything within my power to avoid hurting them:

"This is not the way to write maintainable code, should I tell him that?"
"This estimation sounds wrong, should I say something?"
"We need to stay more hours this week, should I force everyone to come earlier or leave later?"

I kept things to myself, working crazy hours, fixing my teammates' mistakes when no one saw. I wanted things to run smoothly, to keep everyone happy. "If it ain't broken, don't fix it!", right?

Surprisingly enough, my teammates did not appreciate me running around and fixing their code. **The more protective I became, the more disconnected people got from their work.** They lost faith in me as I didn't share anything with them, while all I could think of was how to protect them even further. They did the bare-minimum work required, while I was working for longer and longer hours.

Looking back, I was being selfish. **I judged the situation and acted based on the image I wanted to create for myself**. I thought that if I'd tell them the painful truth, they would stop liking me. Avoiding sharing my real thoughts with them was just an excuse for my fears to damage my image. Thankfully, my need to please didn't last long enough to completely lose my teammates.

The change began when I started to think more thoroughly about my **responsibilities** as a manager. What's in it for them to be a part of my team? What will I teach them that is truly unique and valuable?

To me, teaching someone a new technology or a new programming technique was never good enough.

I wanted to teach them how to deliver products. I wanted to teach them how to ask the right questions, how to look at the bigger picture, how to prioritize tasks and risks, how to build trust by creating visibility into their progress, and so on.

My goal was to help them grow and become great engineers. I knew that it would not be easy, so I stopped looking for easy shortcuts to get there.

I realized that while it's never okay to act like an "asshole", it's probably unavoidable to sometimes feel like one. **Sharing harsh feedback is really hard** on both sides, yet it's a part of our responsibilities, if we really care about growing our teammates.

We need a way to break emotion from behaviors, so we could judge ourselves based on the outcome we're looking for. I would like to go over some guidelines I've used to challenge and confront my teammates from a place of personal growth and genuine care.

Having a few more tools in our arsenal, could help us deal with the stress of sharing feedback and leading our teammates. This is what we're going to cover now.

Behaviors and feelings

Dick Costolo, Twitter's CEO, uses the term "The Leader's Paradox" – as managers and leaders, we need to care deeply and thoroughly about our people, while not worrying about what they think of us.

We need to figure out a way to push our team towards the right direction. It means we need to be mature enough to contain the pain and disappointment when things aren't going as expected. But before we can act as leaders, we have to understand both the power and risks of putting ourselves in our teammates' shoes.

Empathy and sympathy

There is an important distinction between these states, one that we should be well-aware of.

Empathy is noticing someone else's situation or perspective. It means that we get how they analyze the situation. It does not mean we agree with them, or share their feelings. It does not mean we have to solve their problems.

Sympathy is empathy plus *feeling* the same way as the other person. It's more than noticing or agreeing with one's perspective, it's feeling *as if we were that person*.

Let's say one of our teammates is not performing as expected. Empathy will put us in a place where we could analyze the situation from their side: maybe there is a misalignment of expectations; maybe someone or something else was failing them to get the job done on time; maybe they have hard time at home. Sympathy on the other hand, will trigger memories from our own past failures. We will find ourselves justifying behaviors we normally wouldn't, because we remember the pain of our own failures.

Sympathy might lead us to act as if we were the ones to fail. This may increase the chances of us judging the situation from a subjective point of view. Sympathy turns the focus on us instead of focusing on the situation and the teammate.

Our teammates expect us to always act from a place of empathy, not sympathy. To really excel at their work, they need our objective opinion.

The "anti-asshole" checklist (or: making sure I'm not acting selfishly)

In order to judge my own decisions, I had to create a simple way to see if I acted from a place of sympathy or empathy. While I hated the feeling of sharing "bad news", I reminded myself that my teammates expect me to do my best to push them further.

Here are a few questions I've asked myself, after making a hard decision or sharing some harsh feedback:

- **Did I show empathy?** The simplest way to do so is to reflect the situation as we see it, e.g. "The feature wasn't ready on time and we had to delay the entire release. I saw that it was hard for you to get the commitment and cooperation from the Product Team. I also understand that you weren't the one to set the deadline."
- **Did I clarify my expectations?** e.g. "I expect you to raise a flag earlier, if you believe that you're not going to meet the deadline. I want to see you more communicative with the Product Team, even if I'm not available. Earlier sync with them could have reduced the chances of delay on their side. Finally, if you feel the deadlines are wrong, I expect you to ask for help and make sure I am aware of it."
- **Did I practice what I just preached?** It is imperative to demonstrate leadership by practicing it. If we're constantly late with our own tasks (or meetings), we can't ask our teammates to keep their deadlines. If we're bad with communicating dependencies with other teams, we can't really expect that from our teammates. If they see our own boss chasing after us for

status updates, we can't expect them to be active and create visibility into their own tasks.

If I answered “YES” to each and every one of the questions in this checklist, then I knew I acted from a place of growth. It didn't make it easy, at least in the beginning, but it helped me to sleep well at night.

When we act based on our deepest beliefs for what's right for our teammates, when we lead by being forthright and clear, then it's easier to deal with the rest.

While this may feel as if we're losing our friends, judging our actions using these 3 questions slowly reduces that pain. It makes us feel more confident, as these decisions represent our true self, and most people respect that.

If you want to achieve anything in this world, you have to get used to the idea that not everyone will like you.

— Simon Sinek

---

4 common pitfalls to avoid

Failing to publicly share our own lessons learned

Instead of hiding behind our own failures, what if we could share it with our teammates?

Acknowledging our mistakes and sharing our progress creates openness and trust. It shows we are open to feedback, that we are willing to share. It shows that we too are learning how to get better at this role. The medium itself is a matter of taste, but here are a few ideas:

1. Email to the team or company – "My lesson learned from yesterday's deployment mess"
2. A blog post – "5 things I should have done before committing to open-source project"
3. A tech talk in some public event – "Lesson learned applying Continuous Deployment to mobile apps"

The guideline here is to be honest about it. We should share the reality as we grasp it, what happened and how we believe we should handle it next time.

Forgetting to summarize and follow up

Sharing feedback doesn't end there. Improvement takes time. In order to make this process effective, we need a way to track this improvement over time. After we provide feedback and listen to how our teammate responds to it, we should summarize the conversation and send it to both of us via email. The idea is to capture our feedbacks and how we understood the other side and keep it so we could later on follow up on that during their next feature, our next 1:1 meeting or during the performance review.

I used to send a summary of my conversations with my teammates, including my concrete examples and recommended steps. Then, I added **how I understood the other side and the next action items** we agreed on. I asked them to **reply back to this email** with their own feedback if I got it wrong or they have anything else to add. That email was archived (I had a label per teammate), and used as a way to congratulate them on their progress or to figure out together why a certain pattern keeps emerging.

Many managers are afraid of this approach as they feel it may lead to "keeping score". This is, in my opinion, a dangerous pitfall and the reason I've created the checklist above. Effective feedback requires concrete examples, it requires opportunities to try again and track our progress over time.

Using the wrong medium

Picking a medium to provide feedback should always **fit our own style as managers and our teammates' style**: it's a matter of personality, maturity, openness and trust.

I believe that 1:1 meetings should work best if both manager and teammate are introverts, or the level of trust between them is still low. It would also work best if the team is still new and trying to figure out how to work with each other. At this stage, the idea is to get comfortable to the notion of sharing feedback and getting better at it.

In mature teams, we often see more openness in terms of conducting a team retrospective and using 5 whys to reach the core problem. This is a maturity level in which we can also ask our teammates to write about it and share it – may it be at the company's level or with the world.

A couple of great examples: Etsy's Blameless PostMortems and a Just Culture, Buffer's Lessons from Buffer's security breach.

Delaying feedback

We should be fully aware of what it really means every time we say "oh, she's too busy right now, I'll talk with her tomorrow" or "he's having a hard time, it can wait".

Every time we're delaying hard calls or honest feedback, we make an active decision that our teammates could see and learn from. **Not making a decision is equally important and explicit as making one.** The fact that we delayed the conversation doesn't mean the situation didn't happen. On the contrary, it means it happened and we decided it wasn't important enough to take an action.

It also makes it much harder to be helpful and provide constructive feedback as the situation is no longer fresh in our memory.

What started as a "justified" call may become a part of our team's DNA. Suddenly, people will feel uncomfortable sharing feedback and challenge each other. Once it's ingrained, eliminating this behavior will be much harder.

We also have to keep in mind that **bored people quit**. Engineers want to improve and get better over time. They want to learn new techniques. They want to tackle harder problems. They want to gain the respect of their peers and the organization they're a part of. They enjoy new challenges, they appreciate different solutions.

It's our responsibility to help them grow. If we're not giving our teammates feedback and challenging them, they will become unhappy (and bored) and will eventually leave.

Task List:

So, what should you do next?

- ☐ Print the "anti-asshole" checklist or create a better version of it. Have it available, as a reminder for your responsibility for your teammates.
- ☐ Write a list of things you should have said to people in your team but waited with it until now. Set a meeting with them (1:1), and use that time together to push them forward. Use the checklist if you need some mental support.
- ☐ On your next 1:1 with your boss (or over lunch with your peers), ask for their opinion on your empathy versus sympathy levels. Couple of questions that can help:

  "Do you think I'm too protective of my teammates? If so, can you give me a few examples?"

  "Is there some feedback you believe I should have given to one of my teammates?"
- ☐ Clear 30 minutes to sit down and write your own method of challenging your teammates. Share it with people you can trust, so you'll have more ways to drive new ideas and experiment.
- ☐ Watch Dick Costolo Warns Against Trying To Be Liked video
  (Time Investment: 6 minutes)
- ☐ Read Michael Lopp's Bored People Quit
  (Time Investment: 5 minutes)

Track your progress online!

I have completed this session

LESSON 4

Teach how to get things done

Teach how to get things done

THIS LESSON IS ABOUT:

1. Show how to get things done using extreme transparency.
2. Leverage peer pressure as a teaching technique.

Time investment: 15 minutes.

Motivation:

"Lead by example" – one of the most non-actionable advices of our time.

This lesson is about breaking down this common advice into an actionable plan. A great way to amplify teaching is by showing someone else how to get things done, rather than telling them what should be done.

One way I found very effective to show my teammates how to plan, build and deliver products was by occasionally pairing with them, working together on a feature. It took me years of trial-and-error to understand that it's not only about doing the work; it's also about constantly reflecting our thoughts and reasoning while working together. **We should make sure the "why" is as visible and obvious as the "how"**.

Let's get started.

Extreme transparency and explicit dependencies

Extreme transparency is a concept used as a way to make your entire company transparent to all of your employees, to increase their ability to drive decisions as

they have the full context to operate in. Leading our teammates using extreme transparency should have the same state of mind.

Let me use an example to show what I mean: suppose we are developing a feature much like Facebook's NewsFeed, and we want to support filtering of results based on "person" ("Hide all posts by Joe") or "event type" ("Hide stories from Joe's birthday"). While this example is based on my engineering background, I believe you can easily adjust it to your own expertise.

"Managers" should exist for only one reason. To help those who work underneath them be successful and succeed.

— Marc Schiller

Reducing risks (and waste)

Releasing smaller chunks to learn about real usage as soon as possible

The first thing I would do is to sit with my teammate and think of ways to provide value as soon as we can. After all, no one wants to develop something which eventually our users (or customers) will never use. Some ideas I would bring to the table:

1. Can we break the project and release it in phases? For example, can we release just the filtering by people for the first phase?

2. Can we release it to just a small subset of our users so we'll be able to see usage before we commit to larger implementation? This may lead to easier (faster) implementation, as we don't need a scalable solution from day 1.
3. Does it have to work for all platforms or can we start with one platform (e.g. web or mobile)?

We need to show our teammate how we communicate these questions to the Product Team and how we keep other teams (QA, Support, Operations and other technical teams) in-sync.

Scale and performance – always have a backup plan

The immediate risk – slower response time can make the entire homepage more sluggish and hurt users' retention (how many times users get back to the site). This can drastically hurt our business.

So, imagine that our code is out there, how confident can we be in the solution? Did we collect some statistics regarding the number of events a user will see on average? Do we have some estimation for the number of filtered events on average? Can we easily measure the addition to the loading time?

We should be very clear about offering a plan to test for performance as early as possible. We should also **prepare a few alternatives in-case of disaster**. For example, can we pre-calculate some of the results in-advance? Can we set a timeout interval for the calculation? Can we easily turn off the feature?

At this phase, **we do not need to implement anything, just have it as an option**. This will increase the organization's confidence in our abilities to deliver and recover quickly if the unexpected occur.

Planning

Breaking a feature into smaller tasks to increase deadline feasibility

When working on a feature, in most cases, the estimation can be intentionally "crude". No one can really commit to "4 working days and 3 hours" in advance. As long as tasks' estimation remains in days, it's easy for things to get out of control.

We are all familiar with the vicious cycle: "When will it be ready?" -> "Tomorrow" -> "When it will be ready?" -> "Tomorrow" ...

The smaller the tasks are, the more accurate our estimations will get. **This helps to build confidence in the feasibility of the deadline we're communicating**.

I've always tried to break features into a list of 1-3 hour tasks. I believe that we should always consider investing more time in planning and breaking things to smaller pieces than rushing to execution. While it may feel extremely difficult at first, it's an art worth learning, thus worth teaching.

Execute: leverage peer pressure

Create explicit dependencies between tasks

At this point, I would start to split tasks with my teammate. **While it may feel counter-intuitive, I would try to create explicit dependencies between our tasks**. I would ask her which kind of tasks she prefers to take – for example, she focuses on the backend work while I take all of the front-end code for the Newsfeed. We would create together some interfaces and use dummy data so I could work on the front-end without waiting for her to complete the backend. Before splitting up, we would set a time to meet again during the day to see if we can remove some of the dummy data and replace it with her real implementation. **Short iterations are the key here**.

Working closely together while having explicit dependencies in place, creates peer-pressure for both of us. It motivates us to keep the deadline as we are waiting for each other. When we meet again, it's exciting to replace the dummy code with real code; we can immediately see some of the bugs, manage to do some code-review and pair-program if something unexpected happens.

Practice what you preach, and let them see you in action.

Constantly communicate internally and externally

It was important for me to behave as I expected each and every one of my teammates to behave. I reported progress, I made sure everyone are in-sync about the deployment of the feature, I made sure we've got the design and test-cases ready. When things got stuck, I was clear on what's holding us and what it means in terms of our final deadline. I've said "No" to some meetings I thought people could run without me and stayed late if needed. All to keep the deadline I've committed to.

Support, bugs and documentation as first-class citizens

Yay! Our shiny new feature is now live in production. All is done, right? But what about fixing bugs in production, or helping the support team to solve some urgent calls? What about writing and adjusting the documentation?

If we believe these tasks are as important as writing the code, we should treat them as such. Continue to demonstrate the type of behavior (responsive, helpful etc.) we believe is right. We should split the work, may it be bug fixes, support tickets or documentation. Our work as managers doesn't end at delivering the feature to production. This is also a great way to earn the trust of other teams in the organization and create an alignment to what matters most – our users.

While the nature of the work can be different, the state of mind remains the same: we teach by behaving in the most transparent way possible.

Retrospect & Delegate: can they now teach?

Scaling our team to a point where we will no longer be a bottleneck requires us to also delegate the techniques, and provide the opportunities to let someone else practice their leadership skills. Once we feel someone from our team is ready to practice this technique with another teammate, we should help fostering that opportunity. We should be there to guide and provide feedback where needed.

This is why it's imperative for us to conduct a 30-60 minute retrospective after such session. Are they ready to teach someone else? What's missing? What can we do to fix that?

Task List:

So, what should you do next?

- ☐ Pick one or two features you can join to demonstrate the behaviors you expect to see, by using extreme transparency and explicit dependencies.
- ☐ After you complete a feature using this technique, conduct a 30-60 minute retrospective with your teammate. Ask her – what did she learn during this process? What does she think is still missing to make her more effective? Which parts did she enjoy most? Which did she hate?

  Take this as an input for your next attempt, until you master this technique.
- ☐ Once your teammate feel comfortable with this approach, help her find a feature she could practice it with someone else in the team. Ask her to share what she learned during this process and how the teammate she worked with experienced it.

  Help her to master this technique.
- ☐ Read Ryan Tomayko's post
  Show How, Don't Tell What - A Management Style
  (Time Investment: 6 minutes)

Track your progress online!

I have completed this session

LESSON 5

Delegate tasks without losing quality or visibility

Delegate tasks without losing quality or visibility

THIS LESSON IS ABOUT:

1. Must, Delegate and External: figuring out our responsibilities as Engineering Managers.
2. Using clear expectations to define the behaviors and outcomes of delegated work.
3. The end goal of every delegated work: what to aim for?

Time investment: 20 minutes.

Motivation:

What should we do with our time in order to build a highly effective team that enjoys working together?

This is the first question we should ask ourselves as Engineering Managers. Without an answer, we disrespect the scarcest resource we cannot get back – our time. Like any other role in our company that requires a deep specialization, becoming a great manager and leader for our team requires a lot of trial-and-error. It requires putting our "soft skills" to use, and that takes time. Jumping to get another task done is tempting, but are we the only ones who can do it? And even if so, can't it be taught? Otherwise, when will we have time to pick up our heads and start to invest in those tasks that no one else can handle?

"Crap, I've missed 1:1 with my teammates for 2 weeks in a row."

The real challenge in letting go and delegating tasks to our teammates, is dealing with our fear of losing control.

We all have our unique style approaching a given task. There are questions we tend to ask before we even start, there is a certain way we break tasks into smaller steps and execute them. We learned to appreciate the need of keeping everyone in the loop and raising flags in case the original plan changes. This is why we tend to avoid delegating "Maker tasks" to our teammates – we fear of losing that quality and visibility.

Dealing with this fear of losing control requires a lot of attention, patience and confidence from us. Delegation means we have to set expectations for our teammates, trust them enough so we could provide them opportunities to practice, be there for them with constructive feedback and allow them to try again if they fail. We will also have to protect them from the organization's natural stress for speed.

Delegation is also a great platform to teach our teammates how to deliver high quality product, as it puts the focus on sharing knowledge and expectations very early in the process.

I would like to share two frameworks that helped me to deal with this challenge. The first one would help you to better tell apart what you should own versus what you should delegate, and the second to delegate responsibility to your teammates while building trust with them along the way. Let's get started.

The responsibility list: Must, Delegate and External

It was books such as Getting Things Done where I've learned that in order to make better decisions and find my own balance, I first need to clear my mind and write all of my thoughts down. So, let's do just that and start with a quick memory dump.

Where do we stand today?

Here is a simple template I've used to capture my beliefs regarding which responsibilities should I own:

	A	B	C
1	**Must**	**Delegate**	**External**
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

What keeps us busy nowadays?

These tasks stand for what we believe is our responsibility today. We should start by writing these tasks under the “Must” column. Such a list might contain (just as example):

- Implementing risky features (infrastructure work)
- Planning milestones and deliveries
- Assigning tasks to my teammates
- Conducting Code Reviews
- Communicating requirements and progress with other technical teams
- Communicating with Product and Business teams to make sure we're on time
- Interviewing relevant candidates for the team

There is no need to pause and wonder if that's the desired output, we'll have time for that later. At this point, we only aim to write what keeps us occupied.

The second step, is adding the tasks **we believe are crucial for our team's success**, but we often not get to do or do very poorly. Again, we should add them all under the “Must” column. Such items could be: “Prepare personal growth plan for each

teammate", "Create vision alignment around our goals as a team", "Have a weekly 1:1 with each teammate" etc.

So far, we've written down tasks we actually deal with during our day, or believe we should be doing.

Next, we should write down tasks or responsibilities we already delegated to someone else at the team, if we have any. For example, in some teams Code-Reviews are done by the Technical Lead, Architect or distributed across the entire team. This time, we should have it written under the "Delegate" column.

From my experience, many of us making that transition from an Engineer to an Engineering Manager start with writing (almost) all of our tasks under the "Must" column.

Is this "must" a must?

Now comes the hard part – once we've captured the current state, how can we decide which tasks or responsibilities we want to own and which we want to delegate?

To deal with it, we can ask ourselves two questions for each task:

1. Does it utilize my unique strength and responsibilities **as a manager**?
2. Does it serve the leader I want to become in the long-run (does it push me out of my comfort zone)?

Tasks or responsibilities which do not fit these two goals should be delegated to other teammates. The output of this process is to simply **mark** the current tasks we've got under the "Must" column, so we could create a plan to delegate them to our teammates.

Once we've marked them, we should also try to come up with candidates we believe should own these tasks. Next to each candidate, we should add what we believe they're currently missing to get the job done. Is it lack of knowledge? Is it lack of experience (have the knowledge but didn't practice it enough)? Or maybe

lack of authority in terms of how people in the team or at the company perceive that person?

Using the examples from above:

	A	B	C	D	E	F	G
1	**Must**	**candidate(s)?**	**lacks knowledge?**	**lacks experience?**	**lacks authority?**	**Delegate**	**External**
2	Planning milestones and deliveries						
3	Assigning tasks to my teammates						
4	**Conducting Design Reviews to all of the implemented work**	Joe	no	no	yes		
5	**Implementing risky features (infrastructure work)**	Debby	no	no	no		
6	Conducting Code Reviews						
7	Communicate requirements and progress with other technical teams						
8	Communicate with Product and Business teams to make sure we're on time						
9	**Interviewing relevant candidates for the team**	George	no	yes	no		

This quick view will provide us the highly needed focus when building a plan to delegate these tasks to our teammates.

An exercise: taking some (virtual) time off

Let's imagine we're taking a one month vacation and we don't have any connection to our team. Will the team collapse? As hard as it may sound, of course not!

Now, after overcoming the shock that we're not as important as we might think we are, we should try to think how our absence will be fulfilled and which teammate will take which responsibility **in a natural way**. This question can help in making the decision of what can we delegate and to whom.

What should I do with “External”?

Managing-up means we will also need to consider our expectations from our boss or our colleagues. For example: do we expect to have 1:1 with our boss? Are there areas in the company we'd like our boss to delegate to us? Is there a certain feedback or tools we'd hope to receive? Do we expect our boss to help with the effort of building trust between our team and other teams in the organization?

We should write down our expectations under the “External” column.

Great leaders know that it's only when they develop others, that they are truly multiplying their impact.

— Mark Miller

Delegating tasks: defining behaviors and outcome

Once we've got a few tasks we believe should be delegated, it's time to create a plan to handoff each task. We should start by explicitly defining our expected behaviors and outcome, and also our concerns – if we believe a teammate requires technical training (lacks knowledge) before being able to deal with the new task or responsibility, it should be part of our plan.

The idea behind this plan is to provide a single page document to our teammate, holding the motivation, expectations and tools to be successful at owning this responsibility. Having a constraint (one page, not more!) will help us to carefully figure out what we care about the most, and create a relatively simple structure into the way we delegate tasks to different teammates.

	A	B	C	D	E	F
1	**Must**	**candidate(s)?**	**lacks knowledge?**	**lacks experience?**	**lacks authority?**	**Delegate**
2	Planning milestones and deliveries					
3	Assigning tasks to my teammates					
4	**Conducting Design Reviews to all of the implemented work**	Joe	no	no	yes	
5	**Implementing risky features (infrastructure work)**	Debby	no	no	no	
6	Conducting Code Reviews					
7	Communicate requirements and progress with other technical teams					
8	Communicate with Product and Business teams to make sure we're on time					
9	**Interviewing relevant candidates for the team**		no	yes	no	

1-pager

How should a “1-pager” like that look like?

The idea is to define only what we expect to see happening and not how we expect them to get there. We want them to execute in the way they believe is right and best fit for them.

There is nothing better than a real example, so let's assume we want to delegate risky technical features to Debby, one of our most technical teammates. Before handing off this responsibility, we should send this 1-pager to Debby. It will be used as a baseline for our conversation.

Example: Taking ownership over our team's infrastructure (Oren <> Debby)

Goal: Taking ownership of the entire infrastructure work to a level where I *don't have to be* a part of the discussion to reach a decision.

What do I want to see happen (behaviors):

- Being communicative – you'll get a lot of pressure from the team to deliver your work on time, as they'll be dependent on it. You'll also get a lot of pressure from the product team, making sure our application can handle the scale. I want to see you remain communicative and open, rather than shutting down and writing code without talking to anyone. When shit hits the fan, I want to see you calm and raising a flag. If you spot new risks or making a risky change, I want to be kept part of the loop. It's perfectly okay to say "no", it's okay to notify the team or myself if there is a delay, just stay communicative.
- Being active – true ownership means that you're driving the car. I want to see you gathering risks, requirements and feedback. I want to see you making sure everyone understands how to use the system and how to change it. I would like to see ideas and initiatives come from you. If you need time to deal with Technical Debt, I want you to ask for that. You have the "authority" to delegate work to someone else, when needed.
- Being aware – decisions and priorities are always set in a given context. I want to see you explain the reasons behind a given priority or time estimation. I'm not looking for a perfect answer, but I do want to understand the context and reasoning you had while making the call. If you're lacking context or priorities from me, I expect you to come and ask for it.

This effort includes (output):

- Conducting design reviews for every critical part of the system.
- Tracking requirements as part of our backlog (including rough estimation and priorities).
- Tracking quality (bugs) and Technical Debt.
- Writing documentation about how to use the infrastructure ("best practices") and how does it work under the hood (architecture decisions, basic flows etc.)
- Sharing knowledge: pick tasks you can delegate (with your support!) to share knowledge across the team.
- Conducting code-reviews.
- Training at least one more teammate to be able to conduct a code-review in your absence.
- Defining and monitoring our infrastructure's SLA and KPIs: up-time, response time, hits/second, health checks etc.

How would you know if you're doing a great job?

Go over the points above. Imagine you had me and our teammates rank every expectation and output we've specified. Imagine we gave you a 1-5 grade per bullet, where 5 is "I truly trust her with it!" and 1 is "She doesn't give it enough attention". Are you happy with that score?

Next action items (for our handoff meeting)

- Show you how I currently track requirements and prioritize them in our backlog.
- Show you how I currently track bugs and where we're standing with it.
- Go over all of the modules and documentation to make sure you are able to take it from here.

- Notify the team of this transition and ask for their feedback and help.

The end goal of every delegated work: what to aim for?

This list can serve us as a visual tool to manage responsibilities and set expectations both internally and externally – to ourselves, our teammates, our colleagues and our boss.

Delegated work done right means you'll gradually feel comfortable addressed as "CC" in emails, until you reach a point where your input is not required to make an effective decision. You'll gradually feel comfortable asking these people to help with rough estimation and long-term planning, without double-checking.

You'll want to be informed rather than be dependent on.

Delegating tasks is a powerful tool to practice our own leadership skills. Are we able to effectively delegate work with our 1-pager? Can we communicate our expectations in a way that both sides understand how it should work? Do we see our teammates grow and enjoy their new autonomy and responsibility? Are we feeling more comfortable to spend more time on key responsibilities we neglected so far?

Task List:

So, what should you do next?

- ☐ Create your own copy of the responsibility list. You can use a copy I've prepared for you as Google Spreadsheet. Write down everything you can think of under the "Must" and "Delegate" columns.
- ☐ Ask yourself which tasks you believe you should delegate. Share this process and ask for feedback on your next 1:1 with your boss or on your next lunch with another Engineering Manager in the company.
- ☐ Go over tasks you currently own and want to delegate – who should you assign them to? What do they lack? Write down the 1-pager you'd give each teammate to help them succeed.
- ☐ Set a goal to delegate at least 2 tasks you're doing today to someone else in the next 3 months. Talk about it with your boss, so he could be part of this journey and provide you with feedback and guidance.
- ☐ Set an **monthly recurring event** in your calendar to go over your responsibility list – Can you think of more tasks you can delegate? Is there a new "Must" task you should deal with?

Track your progress online!

I have completed this session

LESSON 6

Build trust with other teams in the organization

Build trust with other teams in the organization

THIS LESSON IS ABOUT:

1. Why our default efficiency mechanism creates low trust between teams.
2. Building trust with other managers as part of our execution process.
3. 5 tactics to involve our teammates in building trust with other teams.

Time investment: 25 minutes.

Motivation:

Organizations, just like engineers, often use a "best practice" we are all familiar with – breaking our execution team into smaller parts. One of the main motivations behind this concept is to achieve greater organizational throughput: smaller autonomous teams can execute faster by reaching decisions faster as the dilemmas and problems emerge. They can specialize and own certain areas of the product or technology and above all – it's a great way to distribute accountability, as our teams are small enough to move fast yet big enough to tackle big challenges for the organization.

Execution teams are the core asset of every software company. They are the engine and wheels which allow the company to reach its destination, or pivot along the way. **While the functional aspect of these teams is well defined in most cases, it is keeping them aligned that is really challenging**.

Having multiple teams means no more unified values, context and priorities. It means having implicit expectations between different managers in the company, where every manager has a different opinion on why we need to

operate in a certain way, yet expects the others to align without even talking about it.

As we often tend to do with “best practices”, we abuse it. Instead of talking about the motivations and reasoning for breaking our execution team into smaller parts, we simply focus on output and control. This can lead to teams that naturally optimize their own unit of execution, even if it contradicts the sole purpose the organization is aiming for.

How can we steer our company and move it forward if our teams are misaligned or simply do not trust one another to operate for the company's best interest?

How can we grow the company and scale it, if our unique culture and values slowly lose meaning in the larger scope? How can we allow our people to switch roles and teams if teams are no longer aligned with the company's mission?

In order to optimize for business's success, we need to find a way to align our teams’ vision and goals so we can take advantage of this structure, instead of poorly navigate around it.

I would like to start by sharing a few observations that I believe cause teams to work in silos, due to our default coping mechanism dealing with growth. Setting this context will enable us to dive deeper and talk about how to proactively build trust and involve our teammates in this process. Let's get started.

Why our default efficiency mechanism creates low trust between teams

If I were to ask 100 first-time Engineering Managers all over the world “What is your 1st priority as a manager?”, I believe that most of the answers would be around productivity. This is only natural as these new managers were 100% executioners just a minute ago.

I want to look deeper as some of the behaviors we demonstrate as managers are so inherently embedded in us, that without giving a closer look it would be impossible to form new habits and behaviors.

Here is why I believe that focusing only on productivity might harm trust between teams:

Entering a new world with a very limited context and priorities

Being promoted to a management position for the first time can feel utterly amazing. Feeling as if we are stepping up the ladder leads most of us to immediately accept a promotion, even if the context and expectations from us are completely vague at this point. Working without a clear understanding of what really should be our 1st priority leaves one (huge) question open – what should we optimize for?

Most of us work for years to figure that out.

Lacking vision alignment

Barely being able to cope with our everyday pressure to put off fires, we forget why this unit of execution we're now leading was formed to being with. What is our team's vision as part of the company's vision? How can we set it? Is it even our responsibility?

We were never taught to ask these questions, so we keep our heads on our immediate target and **continue to do what we do best – getting things done**.

Different prioritization and lack of context from other teams

Now let's look at this problem at scale. We've got multiple Engineering Managers with mixed understanding of their expectations, trying to communicate work with each other. Things get really messy when, in order to keep our immediate deadlines, we optimize dependencies by making sure we only tell others what we need from them, without even explaining why. How many times have we spent describing an output we need (say an API) over why we need it to begin with?

Need versus want misalignment

As managers, we tend to mix what we need and what we want. We mix what we must get with our nice-to-haves, so everything becomes urgent and important. We can feel the tension in the air, where different managers are fighting for every feature they need to complete as if the company's faith depends on it. At this point, trust is completely broken.

Let's remove dependencies! (Working in silos)

We hate to fight, so our default coping mechanism hearing “No!” every meeting is to simply go around the problem. How many times have we found ourselves doing everything within our power to go around communication problems, implementing the solution we're looking for ourselves just to reach our deadlines without losing our sanity?

In order to build trust, we have to align our teams to start optimizing for value (getting the **right things** done faster) rather than optimizing our local throughput (getting work done faster). It all starts with consistently sharing context, priorities and pains as part of our execution process.

Building trust with other managers as part of our execution process

It takes a lot of time and effort to create a strong communication and openness with other managers in the company. Where should we start then? One way I found very effective was during our management's bi-weekly sync meeting. Not only did I get to explain what I need to get done by others, I was also able to share context and pains my team was having while also leaving a lot of room to listen and get feedback.

Here is the structure the other managers and I have used during these meetings:

My top 3 priorities for the next couple of weeks

I would start by saying my biggest 3 priorities for the next couple of weeks and why I believe they're crucial. It could be a personal issue like getting a new employee to be a part of the team and become productive. It could also be related to a technical aspect we're worried about and want to make sure it's resolved. Thinking about the 3 things that are absolutely crucial for my team to solve in the next couple of weeks helped me clarify our real priorities, before I start to share my dependencies.

Would you change anything?

Many times, what seems important to us may be perceived quite differently by others. Even though not all of my priorities were always dependent on other teams, it often lead to a great discussion and knowledge sharing if someone else in the group experienced similar pains before.

At this stage, I wanted to see if others had different opinions about my priorities and care to explain why. It's a great way to learn but also to build a feedback loop before you dive into the implementation details.

For example, we may choose to say that we want to work on fixing a performance issue that keeps coming up, while others may feel that it would be better to delay it a bit more as we're swamped with urgent support tickets that may cost us a lot of money in our yearly revenues. Maybe performance is one of the reasons for these support tickets, but if not, are you okay with delaying it to help resolving customers' issues?

Share possible holdbacks and pains

What's holding me back? At this point, I would update about my team's availability (who's taking vacation?) and raise other pains my team is currently dealing with, that may hurt our ability to keep our deadlines. Again, it can be related to personal issues of some of my teammates or technical issues – dealing with a spike of support tickets can reduce availability just like a technical issue that prevents my team from releasing new features to production.

Things I know people depend on

Before I share my own needs, I want to make sure I acknowledged what others need from me. In order to do that, I usually sit with other managers before the meeting for a 1:1 and try to gather requirements as soon as I can. The purpose at this point is not to reach completeness during the meeting, but rather to focus on doing my homework before the meeting, and syncing the entire group on what I'm aware of.

Things I need help with

Now should be a good time to clearly communicate my requirements for the next couple of weeks. Optimally, I would say for each requirement when do I need it for, but I would keep track on it once I get to hear the other managers in the room talk about their own challenges. At the end of the meeting, I would write points next to each requirement to see that it was addressed.

Things to reconsider

After the meeting was completed, I would go over my list and make sure that all of my requirements are acknowledged during the meeting. If it wasn't the case and I couldn't get someone's approval on my requirement, I would grab that person later and see if we can figure out a date for it without the pressure of the entire room. If it wasn't possible, I would start making the trade-offs of delaying the requirement and communicating alternatives with our Product and Business teams, while keeping that other manager in the picture. It's important to refrain from pointing fingers. We live in a world of trade-offs and with limited resources. What can we do to optimize for value? Can we push other requirements aside? Can we simplify the requirement so it would fit (for example: breaking it into smaller deliveries)? Maybe start thinking to hire more people for the team?

Using this structure helped me to make sure I communicate both the work and the motivation behind it. It created some vital visibility into what's going on in my team, while also left a lot of room to get feedback on my planning and enabled cross-team cooperation without forcing it.

A team is not a group of people that work together.
A team is a group of people that trust each other.

— Simon Sinek

---

5 tactics to involve our teammates in building trust with other teams

Once we start building better communication with other managers, how can we build trust between our teammates and other teams in the organization?

The idea, like before, is share not only the technical issues but also the pains each team deals with. We should build a common ground to share our goals so we could relate and even help out when we see the other team struggling. Here are a few ideas that could help with this effort:

A simple "Thank You!" email

During one of your team's meetings (e.g. Sprint Retrospective), try to come up with one individual from another team that helped the team to achieve its goals. At the end of the meeting send an email with a simple "Thank you!" (and a funny geeky photo, of course) to that person and CC her boss and your team. A simple recognition is a powerful motivator.

Internal tech talks

Setting a monthly internal talk where each team presents a single **pain** they're having and how they deal with it, can create a great discussion around that topic. Sharing technical pains tends to create empathy between teams as we're no longer working in silos, where problems and solutions are kept hidden. Many times I've seen people from different teams helping each other out by talking after the session and offering a few more ideas they could try. Put smart people in the room and let them talk about their pains, they'll figure it out.

One side benefit of these internal talks is that they also serve as a great practice for the teammates before they give a public talk. It's great for their personal brand (so people outside the company will know them), and for our Inbound Recruiting efforts as a company.

Cross-team exchange program

There is no better way to fix communication issues than by working closely with other teams. If we're dependent on another team to get our work done, we can send someone from our team to theirs to understand how they operate or even to implement our feature in their codebase. We can also have someone from the other team to spend 2-4 weeks with our team, to better understand the problems we're facing every day.

Exchanging teammates for a few weeks is a great way to produce better understanding, so at the end of this time each team has someone who could better explain the constraints, working methods and end-to-end business value.

Pizzability

This is merely a codename for a small tradition of eating pizza and watching usability videos of our customers using our product. You can switch it with Hamburgerability or Pastability. No matter if the team builds the product's interface or the backend which operates it, sitting down with other teams and watching how our product is being used could create better understanding of the value we produce. It's no longer tracking numbers, but rather seeing how real users use our product. It's a great way to create a bond between teams – by

focusing on the problems the user is facing instead of the daily problems and solutions we're working on.

Invite a (rotational) teammate from another team to our team's Design Reviews

Another way to learn about how other teams operate and learn better about their pains is to participate in their technical meetings. One small trick we can do is to send someone else every time so our teammates will slowly get to know how it looks like on the other team. There should be solid trust between the managers to allow such initiative to flourish, but it's relatively “cheap” and effective way to constantly learn more and build better trust between our teams.

Task List:

So, what should you do next?

- ☐ Take an hour and write your answer for this question: “How can I help in building an organization where the parts in it trust each other and enjoy working together?”. Share it with people you trust, may it be your boss or one of your peers.
- ☐ Even without a sync meeting between managers, you can still trigger a conversation over email every 2-3 weeks (or every Sprint, if you're using Scrum):
  - ☐ Write down your 3 top priorities. Ask for your boss and colleagues’ feedback on it.
  - ☐ Specify your known holdbacks: vacations, production issues, personal issues etc.
  - ☐ Specify the tasks you know people are dependent on and when you believe they'll be ready.
  - ☐ Specify tasks you're waiting for and when do you need them for. Mention that if someone believes she couldn't make it on time, she should reach out to you so you could sit together and figure out what to do next.
- ☐ Find at least one way to start working on building trust between your teammates and other teams in the organization. Internal technical talk or “Pizzability” works well even if trust is not there yet. It takes a lot of time to make that connection, so the faster we can start the better.
- ☐ Try to add a few more ideas/tactics to involve your teammates in building trust with other teams. It's a powerful tool to have and it can be a great discussion with your peers on your next lunch together.
- ☐ On your 1:1 with your teammates, try to figure out if they trust other teams they're working with to keep their deadlines. If not, try to figure out why they have that feeling: is it based on previous deliveries? If so, which? Write it down and talk about it with the other maangers to figure out the next step. Many times, it's only a matter of having a good **external communication** – imagine that someone from Team A drops a feature that Team B needs, due to an urgent support issue. Without communicating this priority change to Team B, trust will be decreased even if it's a justified priority change.

- [ ] Read "The Two Sides of Trust" to learn more about the difference between transactional trust and relational/emotional trust
  (Time Investment: 4 minutes)

Track your progress online!

I have completed this session

LESSON 7

Optimize for business learning

Optimize for business learning

THIS LESSON IS ABOUT:

1. Developing a culture which values business learning over building.
2. The problem with optimizing solely for either work efficiency or throughput.
3. Optimizing for value: how to use our product's lifecycle to drive optimization decisions over time.

Time investment: 25 minutes.

Motivation:

During my time as a manager, I was constantly struggling with figuring out the best process to make my team and company more productive. I have read books about Scrum, Kanban, Scrum-ban and pretty much any other Agile methodology out there. My pool of options got even bigger thanks to the Lean Startup movement, and the constant talks on Continuous Delivery and how it's applied in various startups and (relatively) big companies such as Flickr and Etsy.

When looking at team level, what should we optimize for? Should we focus on reducing the time it takes to release a single feature to production? Maybe we should look at it in terms of our ability to release more features in a predefined time-frame? Maybe we should focus on eliminating future work by making sure that we have a solid infrastructure in place today? What if we want different optimization for different features?

"Premature optimization is the root of all evil" *– it is the same logic we should apply when* we are optimizing our process.

Up until recently, I was trying to extract the most exciting (aka "it's cool!") parts of each methodology, instead of asking harder questions regarding the current state of the product and business. I was optimizing for the sake of optimization.

It was a short and completely unrelated marketing lecture by Dave Mcclure called "Startup Metrics for Pirates: AARRR!" that changed the way I'm looking at processes. Listening to Dave defining the 5 steps each company is going through, was the first time I've encountered such a simple yet powerful language to figure out the current state of the business. This language helped me to understand how different methodologies would better fit each step in the product's life-cycle.

Even if we're not the one in-charge for the process inside of the company, leading others means we have to represent the motivation behind these decisions. The deeper we'll go with understanding the pros & cons for each method, the easier it would get to provide clear and accurate answers. Also, while we sometimes cannot control the process used by the entire organization, **we can experiment with our team's process, as a way to increase the value we provide to the organization**. The biggest changes usually start with a small experiment.

My goal with this lesson is to introduce a few concepts we could apply to make sure we're optimizing the right thing at the right time. Let's get started.

Developing a culture which values business learning over building

No matter what we're currently trying to optimize for, it's important to first figure out the business value from the features we're developing.

As an execution team (everyone involved in releasing the product), our job is not only to appreciate well-crafted software, but also to understand if it makes sense to invest so much in every step of the way, may it be a new feature, an improved infrastructure or a re-design. Here are a few questions we should ask ourselves more often:

1. Will we know when and how people hear about our product?

2. Will we be able to understand if and how they use our product?
3. Can we change things quickly enough to improve our product?
4. Can we measure the quality (usage) of our changes?
5. Can we reduce amount of work, and test for need earlier (think MVP or POC)?

Using these questions to set context and priorities can help our execution team to be more passionate about problem they're trying to solve, and the process they're using to figure it out, as opposed to being passionate about their current implementation.

Don't fall in-love with the solution you've built, but with the problem you're trying to solve. From my experience, the people I enjoyed working with most were ones who follow their intuition and adjust by experimenting, rather than rejecting the unknown and optimizing the irrelevant.

Optimizing for efficiency versus throughput

First, let's start with a quick explanation: when optimizing for efficiency, we aim to reduce the number of "work units" (time, resources etc.) required to **complete a single task**; when optimizing for throughput, our goal is to provide as many completed tasks in **a given time period**.

In some ways, you can compare it to optimizing for a single person (assuming one can do everything) versus optimizing for 10 people with different kinds of expertise. Throughput will hugely benefit from batching and preparing things beforehand (communication, requirements etc.) while efficiency will be hugely damaged from it.

Some processes made this distinction clearer by setting the focus on what we measure – Using Kanban, we measure the time it takes to complete a single feature while measuring the entire pipeline. The focus is on the task at hand and how each step in the process can change the overall time it would take us to deliver that task. Scrum introduced the concept of predefined time-frames (i.e.

Sprints) and how many features you can achieve per that predefined time-frame over time

Simple eh? As it turns out, reality is much more complex than that.

Figuring out the impact of our optimization decisions over time

Efficiency and throughput are not necessarily correlated. For example, let's take a look at Automated Testing:

1. Automated Testing adds time to each feature – this means our **immediate efficiency may get decreased**, as we need to provide the automatic tests in addition to the feature. **In terms of throughput, Automated Testing can increase it** as this process encourages us to spend some more time testing each feature, thus having fewer bugs discovered late in the process by QA.
2. Automated Testing reduces the number of feedback loops between development and QA – having some solid tests in place may lead to a smaller number of hours required by the testing team to verify the feature's behavior. Communication can get pretty expensive, so Automated Testing can decrease the time to deliver each feature and so **increase efficiency**. If communication is extremely expensive, the addition to each feature in terms of testing can be smaller than the investment in communication and pushing back the feature.
3. Automated Testing reduces the number of hours needed for an on-going "regression testing" – assuming that these tests will continue to run for future development, we have a system in place to verify the old behavior isn't broken. This should **boost the throughput** of the team.

Using Automated Testing can lead to both decrease and increase in efficiency and throughput **over time**. What started as decreased efficiency and throughput (investing more time per feature, less time to complete more features), may lead to higher throughput and increased efficiency.

Optimizing for value

Companies that fail to learn and adjust will eventually run out of money, people or customers. It is our responsibility as leaders to understand the context of the company we're part of, if we want to optimize the right thing at the right time.

Figuring out our product's life-cycle using AARRR

In his talk, Dave Mcclure uses "AARRR" to explain 5 phases every company is going through; some of them will happen simultaneously:

Acquisition – Figure out how to bring more users (or customers) to our application. It doesn't matter if it's via ads, having some press coverage or writing legendary content in our blog. Whatever works.

Activation – Make sure our customers would complete a task that would help us convey the value of the application (in other words – the user enjoys 1st time experience). For example, if we're developing an alarm-clock, we'd like to help the user set up her first alarm and show her how it would sound like waking her tomorrow morning.

Retention – Increase repetitive visits to our application. For most applications, without repetitive visits, there is no option to provide long-lasting value.

Referrer – Incentivize your users (and customers) to bring new audience. At this point, the user is satisfied with our product enough to recommend it to others. This can drastically reduce our cost to acquire new customers.

Revenue – Monetize our users/customers.

What I love most about these 5 phases is that it's easy to understand how each step reflects our business. For example, if our retention is pretty low and our visitors do not come back to use our application, then we shouldn't invest money in acquiring more users just yet. Or, if our activation is poor and almost no one completes the task we believe represents best our value, then investing time in improving retention would probably be a complete waste.

While startups are busy figuring out how to reach product-market fit (solid Activation and Retention) with a scalable business model, established companies invest time and effort to increase Revenues by introducing more advanced capabilities, reduce Acquisition costs, incentivize Referrers etc.

As Engineering Managers, it's up to us to protect the team from investing in the wrong place or at the wrong time. While we may feel this is the CEO's or the Product Team's responsibility to make such decisions, it's our job to make these tradeoffs visible. We cannot afford to bury our heads in the code, hoping things will be fine.

When we don't have business certainty, optimize for getting answers as fast as possible

If we're currently working in a team responsible for a product that still figures out Activation and Retention, (I would claim this applies for every product pre-Product/Market fit) we should optimize for **efficiency**. At this point, delivering more and more features will not change the bottom line, except for burning more money and time over things no one would probably use. We need to learn faster what's working and what isn't.

At this point, the best thing we can do is focus on measuring and optimizing the time it takes us to release a single feature (end-to-end) to production, and learn about the usage and impact of that feature. In short:

Have a hypothesis (“this feature will improve Retention by X% due to Y”) -> Release Feature -> Learn -> Rinse and repeat.

Measure each step in the release pipeline

How much time are we spending figuring out the requirements of the feature? How much time does it take to get the design? How much time do we need to develop the feature and test it? How much time do we need in order to solve

critical issues? How much time does it take to deploy and monitor health & usage of the feature?

We should offer ways to cut time where it hurts most. Could we invest 20% to cut 80% of the time-to-release? At this point, we should push ourselves, our teammates and our surrounding, to cut hard into the meat: Can we refine the feature (POC or MVP) so we could **validate** its impact earlier? Can we delay huge infrastructure work, by releasing it to smaller audience? Remember, this feature might not be used at all.

Understand the impact of the feature

We need to make sure we've got analytics in-place, so every feature we're releasing can be measured against the KPI we want to affect. If our feature should be about improving 1st time experience (thus helping with Activation), then we should make sure we'll be able to tell if more people complete their first usage with the application due to the new feature. Without it, we're working in complete darkness.

Kent Beck says it best:

> "*If you can't make engineering decisions based on data, then make engineering decisions that result in data.*"

Word of caution: Technical Debt is less scary than getting out of business

At this point, we might feel compelled to make some decisions which may increase our Technical Debt. Personally, I would be more concerned about getting out of business. As a manager and leader, this is something we cannot simply forget and let others (e.g. the CEO) take care of. If we can buy more time for the business to figure things out, then we should do it. We should act responsibly and communicate things clearly, but optimizing code or scaling up components which may or may not be used is sticking our head in the sand. The same applies for massive infrastructure work and automating irrelevant manual

steps. It's hard to read these lines, I know, but the last thing we want is to invest in things that might be thrown away.

Given business certainty, optimize for predictability

While business certainty is kind of a fluid term, there will be a phase in the product's lifecycle where we'll feel as if things become clearer. It's this moment where sales, initially just a first few, become a common event; where milestones become more and more possible; where champagne is being poured just a bit more often than usual. What's now?

At this stage, most companies focus on growth (Acquisition and Referrer) and re-validating their business model (Revenues) as the company grows. Once the major risk of developing a product for vain is gone, one of the key parameters any team is measured by is how predictable it is: given X resources, what would be the expected outcome?

At this stage, optimizing for throughput can be a great way to measure our ability to deliver predictable results in predefined time-frames with our current team.

Creating a baseline: tracking our team's output in a simple spreadsheet

If you have some definition for timely planning/release (e.g. Sprints), you can use that. If not, pick a time you're comfortable with, say 2-3 weeks, and start measuring the number of completed outputs you're able to deliver. You can use t-shirt sizes, Story Points or anything else to track the team's velocity, for example:

01/01/2013 – 14/01/2013:
2 Large features, 5 Medium features, 4 Small features.

15/01/2013 – 29/01/2013:
1 Large features, 3 Medium features, 6 Small features.

After a couple of months tracking these numbers, you should be able to create some predictability into your plan. You'll be able to commit to a range of feature sizes, based on empirical data you gathered. It won't be perfect, but it won't be "gut feeling" either.

Figuring out and optimizing bottlenecks

Now that we've got some empirical baseline, we can start optimizing for throughput. Here are a few optimization ideas that come to mind:

1. **Resolving dependencies in advance (batch)**: can we sit with other team members in the organization and sync our schedule so things we're waiting for would arrive at time? For example, say we have a team of engineers who need some sort of feature requirements and graphic designs from the Product Team before they can start working. By making sure our plan is clearly visible to the Product Team, we can try to align their output so our engineers would get the spec + design just before they're starting to work on it.
2. **Infrastructure**: at this point, working on infrastructure that could reduce the amount of code needed to develop new features (or adjust existing ones) may be worth it in terms of ROI. The company has a clear understanding of what exactly it tries to solve, which makes infrastructure a force-multiplier, unlike in the previous pre-Product/Market fit days.
3. **Automating manual work**: the team is probably much bigger by now, so it makes perfect sense to invest in automation to cut down repetition and human errors.

It's not a one-way route

Some companies would change their business or try to conquer new areas. This means we cannot stand still, but rather keep optimizing according to the

product's current business needs.

Companies fail when they stop asking what they would do if they were started today, and instead just iterate on what they've already done.

— Aaron Levie

Task List:

So, what should you do next?

- ☐ Figure out your product's current phase. Talk about it with your Product/Business team to make sure you get the picture from their side as well.
- ☐ Try to come up with 3-4 ideas to optimize the value of your team based on your current product's phase. Have a conversation with some of your peers and teammates, so you could brainstorm and prioritize. Some people will resist, usually using the current process as an anchor – "we cannot do it because we're using Scrum with bi-weekly releases". Hang in there.
- ☐ Read Kris Gale's post – Efficiency is not our Goal (improving company's throughput). Kris's post addresses a lot of the issues we covered earlier, from a big company's point of view
  (Time Investment: 5 minutes.)
- ☐ Watch Henrik Kniberg's Agile session (video) to learn how resource utilization can dramatically affect the end result
  (Time Investment: 35 minutes.)
- ☐ Talk about it with your teammates – share your thoughts (and Dave McClure's talk) about what you want to optimize for and why. Set your expectations about their responsibilities of business learning.

Track your progress online!

I have completed this session

LESSON 8

Use Inbound Recruiting to attract better talent

Use Inbound Recruiting to attract better talent

THIS LESSON IS ABOUT:

1. The importance of starting to promote your team and company today.
2. 9 Inbound Recruiting tactics to attract talent.
3. How to distribute and delegate recruitment effort in your team.

Time investment: 25 minutes.

Motivation:

"It's so hard to find great employees these days" – everyone, all the time.

One of the most frustrating aspects of finding a great addition to my team was poor timing, or so I thought. It seemed that the second I started to look for amazing people to join me, the talent pool got "unusually" empty all of a sudden. I kept asking myself "Where do all great people hang out?" and "How come people don't know how amazing it is to work here?"

As a company, we're not only chasing after our next customer. We're also chasing after our next employee. If we're willing to put so much effort in product marketing, why not investing effort in recruiting marketing?

"**What makes us attractive to such people?"**

I started asking different questions: "Which kind of people I want to attract to my team?", "Where are they working now?", "Which meetups do they attend?", "Which blogs or books are they reading?"

Let's face it: great people, those who can really help our company to thrive, are simply not available too long when they seek for a job. In a world where talent is so scarce, we must find a way to rise above the noise.

Dharmesh Shah (co-founder at HubSpot) said something that struck a chord with me: "Culture is to recruiting as product is to marketing". Making their awesome culture **visible**, as they did with their Culture Code slides, made all the difference for HubSpot.

Along the years, I found a few tactics that worked quite well for me. It wasn't always a clear win, but it made an impact. It didn't make hiring easy or simple, but it made us visible and people were talking. This gave us a chance to reach out to people, to start a dialog. When I did something extremely well, people reached out to me. **Sometimes, a short dialog is all you need to hire the best employee**.

This lesson is packed with ideas that could fuel your creativity in building your own "Recruiting Marketing". Let's get started.

Start promoting your team and company today

"We'll make time for hiring better people later. We need to build a product now!"

Short term satisfaction is addictive. Building a brand to help us hire someone 6 months from now is just not as urgent or tangible as releasing a new feature. The problem is that **the urgent is sometimes the enemy of the important.**

When there is no sense of urgency to hire today, and working to build that brand takes precious time, it's really easy to put it aside. No wonder why every time we want to start recruiting, we panic.

The reality though, is that once we set our mind to invest in promoting our team and company to attract talent, there are many ways to achieve it. Even better, it doesn't have to take that much time.

Inbound Recruiting – teach and inspire future candidates

Here is something we tend to forget: we already invest a lot of effort in educating, inspiring and growing the people in the company. We're pretty amazing at it too: we have Code Reviews, Pair Programming, Sprint Retrospectives, 5 Whys, Post-mortems and many other practices to share knowledge and train employees.

What if we could leverage that to teach and inspire people outside of our company?

Build long lasting value: Outbound Recruiting versus Inbound Recruiting

This notion of "Outbound versus Inbound" came from the marketing domain. Put simply, Outbound Recruiting refers to a direct reach from our side to the candidate's. It can be a phone call, a LinkedIn message or simply by approaching someone in a conference and see if we can convince them to join. We're the active part of this dialog. The second we stop being active is also the second we stop getting new people into our recruitment pipeline. Or, as most of us deal with it – imagine the second you stop paying recruiting companies.

Inbound Recruiting focuses on building relationships with future candidates. It works by investing the time in content, talks and projects your future candidates would find appealing. Or, as most of us deal with it – posting more photos to Facebook from our last company's trip or event. But it's more than that. Many people reach out to Google asking to apply for a job there. It happens because Google built a brand of top talent, with an amazingly unique technology and mission statement. People want to be a part of this brand.

Inbound Recruiting has a long-lasting impact. Companies that built a brand over time, still benefit from an unfair advantage of talent available to them: Google, GitHub, Twitter, Facebook, Dropbox, LinkedIn, Quora, Instagram, Amazon, Etsy and the list goes on and on.

"Well sure, they're hugely successful!"

It seems that all of these companies already made it, so no wonder people are drawn to them like magnets, right? But these companies became successful due to their ability to attract talent in their **early days**.

Instagram talked about their unique infrastructure and lessons learned a year before they were acquired by Facebook for 1 billion dollars. GitHub shared how they work as a distributed team more than 2 years before raising $100M dollars to change the world. Facebook pinned job posting at Standford's campus to build awareness and find top engineers for their small startup (crazy eh?) many years before their IPO.

These are just examples. Each company started to build its own unique brand many years before anyone could attach massive success to them. If we're lucky, building a great **brand** will enable us to hire **amazing people** which will help us to find and build a scalable **business**. The best companies out there are hugely successful because they already know that.

"Make something people want" includes making a company that people want to work for.

— Sahil Lavingia

---

9 Inbound Recruiting tactics to attract talent

The goal of a successful Inbound Recruiting tactic is to cause a "wow factor" for our future candidate. Best case scenario would be that this person will sit at home, read our content or play with our project and say "oh wow, that's amazing! I wish I could work there".

Here are a few ideas you could apply today to start building that brand, using the knowledge you already feel comfortable with:

1. Invest one hour a week to answer questions online

You don't have to rush and start a blog. Just be helpful. Pick topics you're feeling comfortable with and invest a full hour to help out at StackOverflow, Quora, GrowthHackers or any other website where you can contribute your time and expertise. One hour a week is a small portion of your time, but the value added over time can be a game changer. Personally, I like using focus booster and set it to an hour.

2. Hide little gems in your product and website

If you want to attract creative people, you have to be creative with the ways you'd approach them. For example, if you're building a web product you can hide geeky jokes fairly easily:

- By using the console.log or simply as plain HTML.
- As a custom HTTP Header, e.g. header('X-Recruitment: We're hiring humans http://site.com/jobs!');
- As a 404 page. My all-time favorite: http://www.masswerk.at/404.
- As a small hidden message inside of your logo: it can be by using your own logo to convey a hidden message (examples), or by injecting a very small vector that only curious people will find. Be playful.
- Use cornify in a secret page in your website or build your own "geekify" concept.

3. Write guest posts about things you've learned or done unconventionally

Most of the people I talk with tend to hold themselves from writing guest posts because they believe what they know is "a common knowledge by now". You'd be surprised how much you can teach and people want to learn. Did you recently tried a new technology and you've got something to say about it? A few tips you can share about building an amazing team? Maybe used a growth hack tactic

only to find out it doesn't work on your type of product? Or, maybe you use Trello in a very untraditional way to manage your project?

Things that seem obvious to you are probably completely new and scary for others. Sharing your experience and the lessons you've learned can be incredibly valuable.

Go over the things you've talked about at lunchtime during the past 3 months. Then seek out a blog you usually read (and hopefully your candidates too) and ask to publish your content. You'd be surprised how common it is, as there is a mutual benefit to you and the blog's owner. They already have the audience, use it.

4. Celebrate birthdays with a twist

Making your employees happy is important, not just for the sake of day-to-day work, but also for marketing. After all, when someone else asks them "where do you work?", you want them to be proud of it. But it's not enough as it won't make you stand out as a company. We need to step our game up.

Occasionally, invest some extra time to build a website (here is what we did at my current startup) or an awesome video to celebrate a teammate's birthday. It's the little stuff that matters most. Share it with the world to get people excited and talk about it.

5. Host a Hackathon around topics you're passionate about

Instead of spending $2,000 on some referrer program, see if you can invest it in hosting a small Hackathon to promote something you are already passionate about. If your team happens to be a huge fanatic of robotics (so what if they're currently developing an iOS app?), then set up a Hackathon around that theme. The same could be applied to many other themes: nostalgic games, acts of kindness, augmented reality etc.

Using a theme could help you to attract people who don't know you yet, but are passionate about that theme.

Then promote the site by using your employees' networks. Hackathons are a great way to introduce potential candidates to the team, and show them how it

feels like to work at your office. Make sure to record the event, the material – videos, photos and products built during that day – are a great content you can use in your blog or even contribute as open-source projects.

6. Give a 30-minute talk

It's time to open meetup.com and look for nearby meetups you believe your future employees attend today. Prepare a 30-minute talk based on a guest post you wrote or a scenario you were facing at work. Practice it internally and keep it short, especially if it's your first time as a speaker.

The real power of giving a talk is that it's easy to reuse in multiple events, so write down 3-4 meetups you think are relevant and send your talk to the organizers. People tend to connect faces to companies, so there is no better way to earn credibility for the company than sharing lessons learned and providing value to your audience.

7. Use side-projects

Side-projects are a great way to create buzz around the people in the company, as it says a lot about your passion to build. Unlike releasing code to open-source projects, side-projects are often easier as they don't require you to invest in documentation or supporting the project for the long run.

For example, you can create a simple website that helps other companies to generate a widget like this (just a mockup):

People could fill their teammates' StackOverflow/GitHub/Quora usernames and it will automatically generate a widget they can use in their "about" or "jobs" page. Once it's up, publish it to Hacker News and LinkedIn, and see if people find it useful. Adding a simple "Powered by [Company Name]" to it would bring a lot of good vibe around your brand.

It can be a tool you're using internally in the company, or even something you've made just for the sake of helping out. Just look at what AppSumo did with How I Got My First 3 Customers or Qualaroo with GrowthHackers, even though it's not a part of their core business.

8. Release internal tools as open-source projects

Can you think of tools you developed internally to get your work a bit easier, that you can push as an open source project? Make sure it appears in your jobs page (this is where future candidates would go), so people will know you're passionate about contributing to open-source projects.

Some examples for outstanding open-source projects include Netflix's Chaos Monkey and Etsy's deployinator (also, check Etsy's open-source projects page).

9. Create a public challenge

If you believe that your best future employees are ones who love to solve puzzles and enjoy a good hack, then prepare one for them. Quora and Dropbox released challenges that lead to great exposure. While there is a big amount of effort to put into this sort of challenges, you can create a relationship with the people who try to solve it.

How to distribute and delegate recruitment effort in your team

Building a recruiting brand takes time and effort, both mentally and physically. The more people we can involve in making it happen, the better chance we have to see it coming through. So while we may want to own the effort, it's imperative to include our teammates in it. They need to know we are willing to invest the time in it, and that **it is a part of the team's responsibility to attract great talent**.

It should start with a simple time allocation. Investing one hour to write answers in various sites can be assigned to a different team member every week. If one of our teammates prefers to get a review before he publishes an answer, we should make it easy for him to get that help from us or another teammate. Even better,

we can make a small competition (if we believe the team would enjoy it) to see how many up-votes people collected each week.

Same goes for investing the time to add some hidden gems to the product – just build it into your schedule. We can set up a 30-minute brainstorming session to come up with ideas and then make sure people could implement some of them when working on their next feature.

Once our team starts to feel comfortable with working on building a recruiting brand, we should assign ownership to these efforts. In your next 1:1, figure out which of the ideas above (or anything else that comes to mind) could be of interest to them to own. Next, ask them to commit to a medium-term goal, for example: "I want to invest 3 hours a month answering about NodeJS" or "I want to give 2 talks in the next 3 months about applying Lean Startup concepts in big companies".

Make their commitment very specific and help them as much as possible to be successful at it. Provide feedback: help them find the relevant meetup group and improve their presentation; see if you could find a blog to publish their content in; help them figure out the MVP of a side-project; get your management's approval in releasing an open-source project. If you believe someone is taking too much on herself, then offer ways to break it into smaller steps.

Like many other aspects of building a company, it's a marathon and not a sprint, so make sure your teammates constantly invest some time in building a recruiting brand, rather than a single big bang effort.

Task List:

So, what should you do next?

- ☐ Consult with your teammates – what would they feel comfortable starting with? Make commitment and track weekly time investments of 1 hour per week to start with. Use your whiteboard to say who's responsible this week for Inbound Recruiting, for example: "Oren to answer 3 questions at StackOverflow about MongoDB or Java". Every time I answer, I need to update it to 32, then 21 and finally 1DONE!
- ☐ Open a wiki page or use the whiteboard to brainstorm ideas the team can invest in, to create better branding. Create some rough estimation for each and ask the team to vote on which they believe will be the smallest to take yet with the highest value for someone outside of the company.
- ☐ Pick one or two suggestions every month and plan time for it as part of your schedule. The idea here is to consistently invest in it, so the team could get used to it.
- ☐ Talk with your boss and with HR (if you have one) – can you invest some of the budget into an interesting Hackathon?
- ☐ If you want to give a talk or help one of your teammates to come up with one, check out http://speaking.io/ to practice your skills.

P.S. Here are some references you can use if you need to convince your boss of the importance of Inbound Recruiting:

http://www.instigatorblog.com/future-recruiting-inbound/
http://mashable.com/2013/09/26/smart-moves-recruit-engineers/
http://moz.com/blog/inbound-tactics-make-hiring-easier-and-more-fun
http://blog.hubspot.com/insiders/using-inbound-recruiting-attract-hire-retain-top-talent

Track your progress online!

I have completed this session

LESSON 9

Build a scalable team

Build a scalable team

THIS LESSON IS ABOUT:

1. The traits of a scalable team
2. Making it a reality: reaching alignment through vision, values and expectations
3. Monitoring bottlenecks and misalignments

Time investment: 30 minutes.

Motivation:

Many years before I was in a management role, I used to practice my "scalable software" craft on a daily basis: building a scalable infrastructure was part of my job description. I read books, went to lectures, applied some well-known practices using well-known tools and managed to fail enough times to figure it out eventually. Failure was immediate and well-defined. If something didn't work, I simply changed it and tried again. My code was tolerant to my skills. It didn't think less of me when I broke it.

Scaling humans is much harder. **Unlike code and architecture, people have expectations, needs, desires, and dreams**.

I wanted to create a team that would be happy to work together, where people knew what is expected of them and were challenged and engaged in what they do. But how do we get there? Where should we start? What is it that makes people happy to work together or being consistently engaged in what they do?

I started to look deeper. I was curious to understand every aspect of this new "human architecture" I envisioned but couldn't put into words. My brain was spinning non-stop.

I started to look at things the team is doing extremely well and things we still lack. I started to ask questions about the business so I could try to figure out how it would affect my team. Even when I wasn't actively interviewing people, I was asking myself different questions about who should we hire next or what needs to be changed in order to tackle new problems.

Along the years, I've come up with a few directing questions and frameworks to guide my thoughts and execution on growth. This lesson should help you to prepare for scaling your own team, by setting the context and providing the tools you can use, even before you start interviewing. Let's get going.

The traits of a scalable team

Building a scalable team means we can adjust our goals to meet new challenges as an execution unit, without losing our unique culture or our ability to deliver.

It means changing direction if the business doesn't make sense anymore, without feeling resentment: “but I've invested so much time in it already, there must be something we can do with it!” It means growing from a team of 2 to 5 without feeling pinned down: “I cannot get things done anymore; all I'm doing is holding hands of incompetent employees.”

Just like scalable architecture – Many things need to happen under the hood to support this growth, but the end result remains the same: a smooth transition into a new reality.

Before we take our first step in building a “growth plan”, it's important to understand the traits of a scalable team. It will serve us as a compass while we execute our plan and make the relevant adjustment along the way.

Alignment of vision

Scalable teams understand the context in which they operate and can **emotionally connect to the grand vision of the company**. They can define their own purpose and their role as a team inside that context and measure themselves by it.

Misalignment of vision will lead to a dysfunctional team, one that cannot create continuous value to the company and sustain changes. When we don't understand the problems our company is trying to solve, it's easy to fall in-love with our immediate results. It's easy to deflect or even point fingers when things move into a new direction, as we were never emotionally invested in it.

Alignment of core values

Scalable teams define and cherish core values they believe are fundamental to **why they do things in a certain way**. It can be regarding the way they communicate, the way they approach validating ideas and implementation, the level of assertiveness and ego they are willing to deal with, what they believe is right in terms of work/life balance, and the way they treat failures or celebrate success.

What if some people in the team don't share the same core values? They will behave in contradiction to others' "true self". Over time, it would trigger a conflict which will undermine the entire team-dynamics we have been trying to build. Just imagine a team of people who believe in honest and open feedback while one of them believes it is not needed, and prefers quick decision making even if it means avoiding the team when making these decisions. Combine it with some ego, and you've got a ticking bomb waiting to explode.

Core values don't have to be curved in stone. If the team feels that one of the core values is no longer relevant, it might mean that it's a time for a change.

Self-balanced

Scalable teams **distribute both functional and growth responsibilities**. As such, not only do people fully understand their own function in the team (e.g. front-end engineer) but also their responsibility and ownership (e.g. mentoring other engineers regarding front-end practices, conducting code-reviews in their area of expertise, enabling others to conduct code-reviews etc.)

Distributing responsibilities related to the team's growth is crucial as it encourages autonomy and mastery for each team member: it helps technical

leads to leave their comfort zone as sole executioners and challenges them to analyze, communicate and mitigate risks. It encourages making decisions based on the better good of the team. It creates healthy expectations of junior employees to practice their versatility over time, while distributing mentorship to other team members.

When people are given purpose (vision), autonomy (decision making and growth ownership) and mastery (functional ownership), they can reach decisions based on their expertise and what would fit best to the team's values. They won't rely on us to get decisions made.

Core values over individuals

Scalable teams understand that reaching an alignment with core values **may lead to losing existing talent due to personal traits rather than technical ones**.

Building a team which works well together requires long-term thinking and the patience to get there. Losing talent will always drastically hurt your short-term productivity; this is why we usually prefer to avoid reaching a decision. Every time that happens the team-dynamics will change, questioning the core values we picked as a team.

Sense of accomplishment

Scalable teams celebrate their victories as they continue to improve and tackle obstacles. The only guaranteed outcome of completed work is generating more work. Without appreciation of effort and celebration of hard-earned victories there can be only burnout. Great teams do not allow themselves to get to that point.

As managers, our job is not to solely own risks, quality and mentorship, but rather to design a process that will distribute them.

Making it a reality: reaching alignment through vision, values and expectations

How can we take these ideas and make them into a reality? The most effective way I found out along the years, is to write things down as we currently grasp them and involve everyone around us until we reach something we feel good about.

Defining the team's vision

I wanted to share some of my own experience in this process, from back at 2007, when I was working at a startup that tried to re-invent search by integrating social information to rank the results. The company's vision was to create a personalized search experience for every human on the planet.

That made me think – How can I make it more relevant to the team's area of expertise? What will inspire my teammates and make them more engaged?

My team was responsible for crawling and parsing the web to build a graph of people and their content, so that the Search Team could use it to rank the results according to the connections between people. We were basically building the database the Search Team used as a service.

So I spent a few hours coming up with what I believed would best represent the team's vision. Once I had it written, I gathered feedback from various individuals in the company and involved the team in it. I've used these two questions:

"Will it move the company into a winning position?"

"Is it big enough as an engineering challenge?"

After a few rounds of feedbacks and adjustments, the team's vision was set:

To build the largest, most informative profile-database in the world.

Defining the team's core values

Once a clear vision is in place, figuring out our own core values is next. As we all tend to appreciate many different traits, in order to really pick the core ones, we should ask ourselves one guiding question while judging each value:

“Is it important enough for me that I'd fire someone who doesn't agree with it?”

Many of you reading this line can hold back and explain why diversity is crucial for building great teams. There is no argument there. There is however, a huge difference between defining the values we believe in and the values we cannot operate without. Core values are non-negotiable, and yes, it may mean losing some people along the way. It's a better outcome than daily arguments. Just think what it does to our teammates to see this clash, from their point of view.

Write your current thoughts on your core values and attach a reason for why it's so imperative for you. It has to be convincing, your passion should be talking. Just to give you a few ideas, here are a couple of core values I've written in previous teams I worked at:

Own it. Never let someone else fix our own mess: we understand that sometimes it's not really our fault, but we don't believe in making excuses and pointing fingers. We believe in getting things done and take full ownership of our deliveries. Above all, we never let someone else deal with the mess we own. Shit happens, and when it does, we'll fix it. This is what builds trust in the organization.

Loyalty to each other above all: we want to be proud being part of the team. That could happen only if we would help each other to be successful, and genuinely care for one another. If someone is falling behind, it's our job to stick around and help out. If someone is kicking ass, it's our job to recognize and celebrate it together. Most chances, we'll see each other on a daily basis more than we see our families. Let's make sure we enjoy each other's companionship.

Using these values can help you for your hiring and firing process. Learning to ask questions during an interview to figure out if they tend to point fingers to explain their own failures, could help you test for a culture fit. Talking about a “crunch time” the candidate was facing in their previous work and how their teammates behaved, can teach you a lot about their core values.

Cynical people would always look at words such as vision or values as empty words. Every time we do not act based on these ideals, means no one will take it seriously. This gentle team-dynamic we've been working so hard to create, will vanish. After all, **we are what we do, not what we say**. We need to act upon these values!

Defining what our teammates should expect of us and the team

What's the purpose of our job as Engineering Managers? It doesn't have to be implicit. We should tell our teammates what they should expect to receive from us and from the team they're part of.

When I wrote to my team, I used these points:

What can you expect of me?

- Providing the best environment to make you productive and happy.
- Share why we do things, not only how we do them. I strive to change our methodologies and tools to better fit our team: stagnation will eliminate our ability to innovate.
- Make sure you've got the full picture: no walls between you and the Product Team / Business / QA teams. I don't believe in hiding the facts or micro-managing your work.
- Getting real and honest feedback from someone who cares about your personal growth. That means I'll push you hard to get you out of your comfort zone (note: you may not like it at times).

What can you expect of being part of this team?

- A chance to solve really hard and interesting problems: crawling, parsing, monitoring and persisting a graph of billions of documents at huge scale.
- A chance to learn and absorb new ideas from incredibly smart people.

- A chance to teach and leave your mark on why and how we get things done.
- A chance to mentor new employees who join us.

At this point, it would be highly valuable to share it with our boss and other Engineering Managers in the organization. Their feedback can give us some more ideas regarding the value we can provide and even provide us with tools to make sure we can deliver.

Defining your expectations of your teammates

Just as important for the team to understand what they should expect of us, we should explicitly say what we expect of them. This time, we'll break it into 2 parts: baseline expectations and personal expectations. The former will include all general expectations we have of any team member, while the latter will be specific for each individual in the team, according to their position, experience and personal goals.

Baseline expectations:

What do we expect to see from each teammate? Here are a few ideas:

- Passion to learn & get better every single day - Agile state of mind – constant feedback, constant improvement, baby steps for large change.
- Passion to teach others, even if it means you'll have less time for hands-on.
- Get it Done –figuring out the requirements, building the feature, making sure we've got monitoring and analytics in place, making sure we can deploy it, reducing risks along the way, communicating with all relevant members on progress. You own your work, not just babysitting it.
- Know when the work doesn't make sense – invest time on things that move the needle. It may be business-related, productivity-related or team-happiness-related.

Personal expectations:

Create a list of all of the people in your team and write 2-3 specific expectations of each one of them, based on their role, experience and strengths.

For example, I've expected every Technical Lead in my team to be accountable for technical risks and technical growth: I've asked them to spend a few hours and prepare a plan to handle these risks. It may be as part of working on other features, writing a proof-of-concept, or anything else that they believe is right. It's their responsibility to come up with suggestions, though we should be there for them to offer a few more solutions and provide feedback.

I've told them that I expect them to take care of technical growth for the other teammates. It means looking out for technical debt, conducting code reviews, improving the team's technical skills via lectures, sharing links to great articles and teaching as much as possible. From them, we don't want to hear "we have shitty codebase", we want them to take action to fix it. We don't want to hear "I'm the only one capable of doing Code Reviews"; We want them to train others to do them too.

We're building a team. That means everyone should have their own responsibilities, both to deliver results while also looking around and helping others. Our Technical Lead(s) can help us building that team, if we'll let them.

If we don't have anyone in our team that we can count on at this point, we should make it our goal to train someone for that role. We cannot be responsible for both personal and technical growth in the long run. We'll be mediocre at best at both.

Explicit expectations create a contract between us and our teammates. This kind of peer pressure is a great way to make sure we are accountable for our promises. We no longer lead by chance, we lead with a purpose. Writing down our expectations is not an easy task; it takes time to reach that quality. Have the patience to iterate on it.

You can't empower people by approving their actions. You empower by designing the need for your approval out of the system.

— Kris Gale

Monitoring bottlenecks and misalignments

In order to keep tabs on our team's progress, we can use these 4 guiding questions to detect bottlenecks and misalignments:

Who is all over the place?

The quickest way to figure out if someone is becoming a bottleneck is watching closely on our teammates and see who's running around to put down fires. In terms of scalability, this phenomenon can get dangerous; the second this person takes a vacation, the team will completely shut down. We need to set our expectations of that person to distribute knowledge and mentor others to get the job done.

Who is currently an expertise bottleneck?

Next, who from our team possesses knowledge that prevents the team from making decisions without them? Or – is there only one individual who knows how to do a specific task? What will happen if that person decides to quit tomorrow?

Just like before, we need to set our expectations of that person to distribute knowledge and mentor others to get the job done.

Who is not building trust with me or the teammates?

We often forget that trust is not a given. The fact we hired someone doesn't mean we completely trust them to do their job. We hired them because we believe they can. Employees who constantly share their status, keep the deadlines and take ownership are ones who actively build trust inside and outside of the team.

We should ask ourselves who in our team is not building that required confidence with us, or with other people in the organization. It's our job, as their managers, to set the expectations straight and help them build that confidence and trust. Talk with these employees, explain how trust is being built and offer ways they can practice it in their daily work.

On your 1:1, ask "what's the worst thing about working here?"

One trick to find either reasons for happiness or productivity blockers, is to ask to point a finger on the worst part of the job. Sometimes, it may lead to an observation on the tools and methodologies we use, while in other times it may raise a personal issue of the way the team operates.

We should listen carefully and try to ask more questions to better understand the problem, rather than quickly try to come up with solutions. The purpose of this talk is to learn more about them. We can use our next 1:1 with them to raise some suggestions and offer ways for them to experiment with different approaches.

Task List:

So, what should you do next?

- ☐ Write down your team's vision. Go over it with your boss, both to get more ideas and to make sure you're aligned. Then, use your 1:1 with your teammates to share your thoughts and get their feedback.
- ☐ Write down your core values. If you need more references or ideas, you can look at Buffer's values (slide 5) or HubSpot's values (slides 7-14). At the end of the day, it's going to be values you and your teammates truly believe in.
- ☐ Write down your expectations, both what the team can expect of you and what you expect of them. For your convenience, you can download (zip file) Luc Levesque's format to write your expectations of your teammates.
- ☐ Share your core values and expectations with your boss, colleagues and (if exists) Human Resources. Get as many inputs as possible, but at the end of the day make sure you honestly believe in what's written there.
- ☐ Create a short presentation and share it with the team. It may feel awkward, but don't sell yourself short. People should feel your honesty and willingness to commit to it.
- ☐ Who's currently all over the place? Sit with them and figure out how you can help out by distributing responsibilities to other teammates.
- ☐ Where do you have an expertise bottleneck? Create a plan to transfer some of the knowledge and assign a task to that expert to teach at least one more person in the team.
- ☐ Who's currently not able to reach your expectations? If not you, who in the team can serve as a mentor and help them get there? Assign them both to work on some of the team's tasks together, at least once a week.
- ☐ Watch Dan Pink's TED talk on motivation at work
  (Time Investment: 19 minutes)

Track your progress online!

I have completed this session

The end?

Our goal as leaders is to truly inspire our people.
To do that, we must allow ourselves to become redundant.

Our mission is to help growing self-managed employees.
To do that, we must set expectations, teach and hold people accountable.

Our output is building a truly scalable team and company.
To do that, we must challenge ourselves to define what it means in our world of values.

This journey never ends; your own path is awaiting.

About the author

My name is **Oren Ellenbogen**, an engineer with a unique passion for people and culture.

In the last 15 years, I've served in the Israeli Air Force, worked for big companies such as Mercury (acquired by HP), and small startups such as Delver (acquired by Sears), Cleeqa and Commerce Sciences. I had the privilege of working with amazing people, serving in different roles such as Technical Lead, Engineering Manager and Director of Engineering.

As much as I enjoy building tools and products, I find myself most fascinated with "Company DNA" - which companies in the world change the way we build software? How do they approach it? How are they hiring people? Which process do they apply? How do they measure themselves?

I enjoy writing and lecturing about these subjects, hoping to provide practical and pragmatic tips people could apply.

Sharing my lessons learned:

- Curator of SoftwareLeadWeekly - a free weekly email, for busy people who care about people, culture and leadership.
- Writing a blog about engineering, culture, leadership, management and startups:
    - Team Lead – here is what your boss isn't telling you, yet still expects of you
    - How To Use Your Unfair Advantage To Create an Unforgettable First Day For New Hires
    - What Dan Ariely can teach us about Software Development

- Lecturing about:
    - "The Leadership Role @ the Agile era" (at Outbrain, Google and ILTechTalks)
    - "12 Leadership Hacks For Building Great Teams" (at Conduit, Outbrain and ILTechTalks)
    - "Lessons Learned Implementing Scrum" (at Agile Practitioners IL)
- Helping out on Quora:
    - What are the biggest lessons first-time managers learn on the job?
    - What are good ways of quantifying code quality?
    - How do you fire someone everybody likes?
    - What are the coolest startup culture hacks you've heard of ?
    - As a manager, how do you handle low morale in your team?