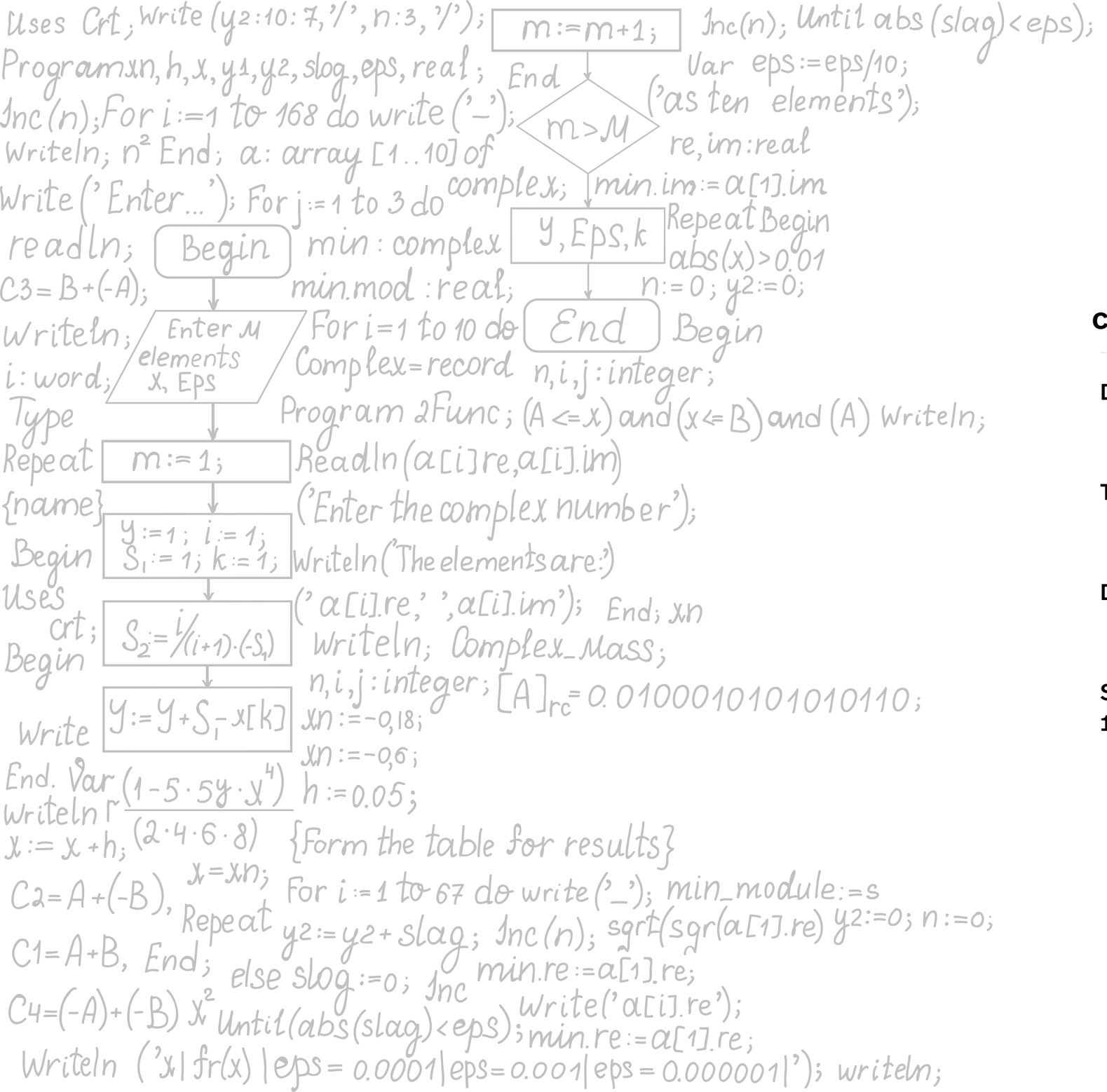
The Dev Team Playbook

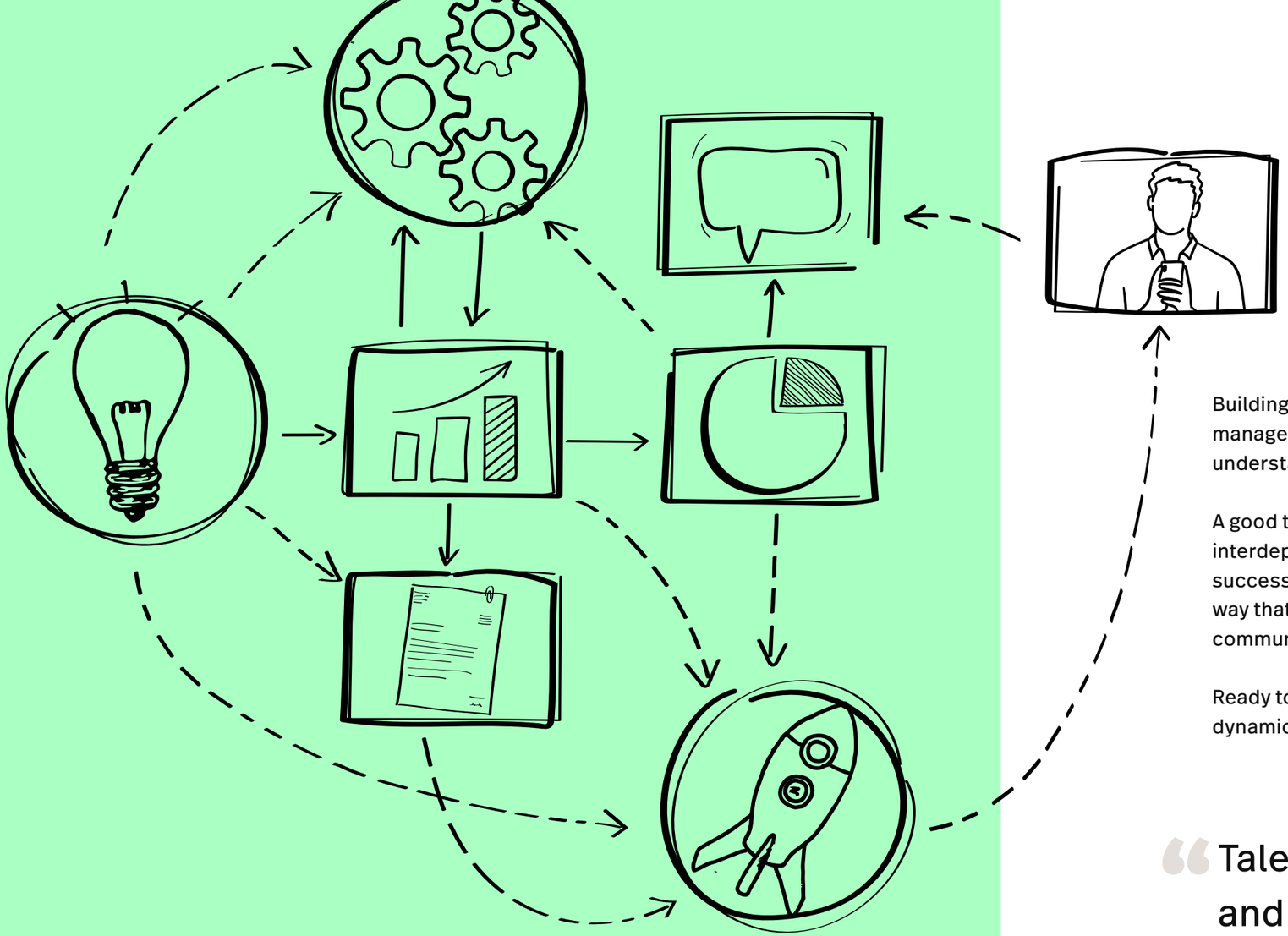
Strategies and maneuvers to help you build a winning team dynamic

Uplevel



Contents

Development is a team sport	1
The human side of team building	7
Deep Work and the art of blocking distractions	15
Strategic play calling for better 1:1s & sprint retros	19



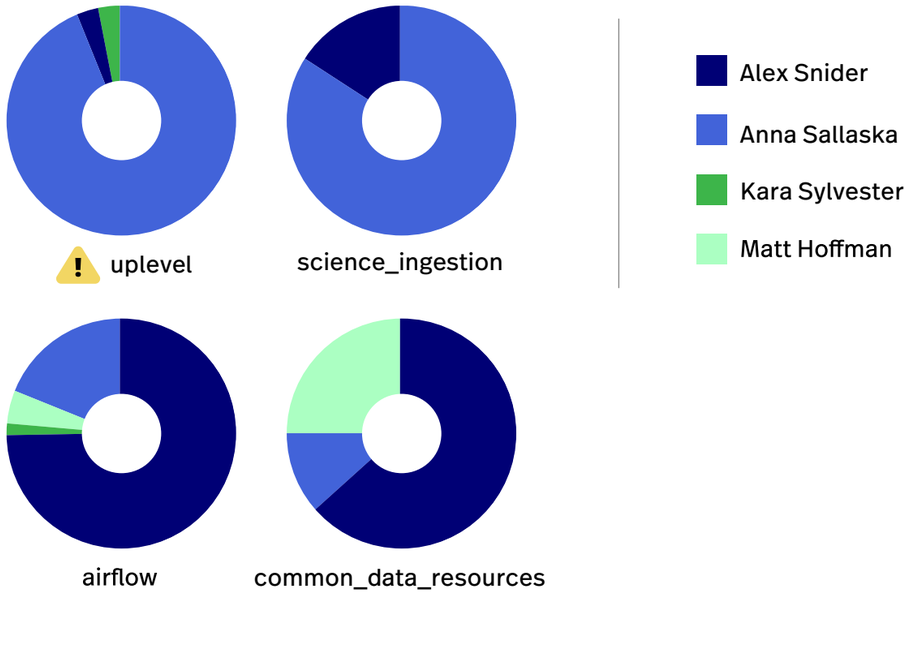
Development is a team sport

Building a great product requires teamwork between all players: devs, managers, execs. More important than org chart hierarchy is a true understanding of the deep value of each role.

A good team can't win without a good coach – and vice versa. The interdependence between devs, managers, and execs is vital for success. It's not only about being your best but also working in a way that helps everyone succeed. It all comes down to engagement, communication, and collaboration.

Ready to play ball? Here's your playbook on building an effective dynamic to lift up your entire team.

> "Talent wins games, but teamwork and intelligence win championships."
>
> –Michael Jordan



Managing knowledge silos

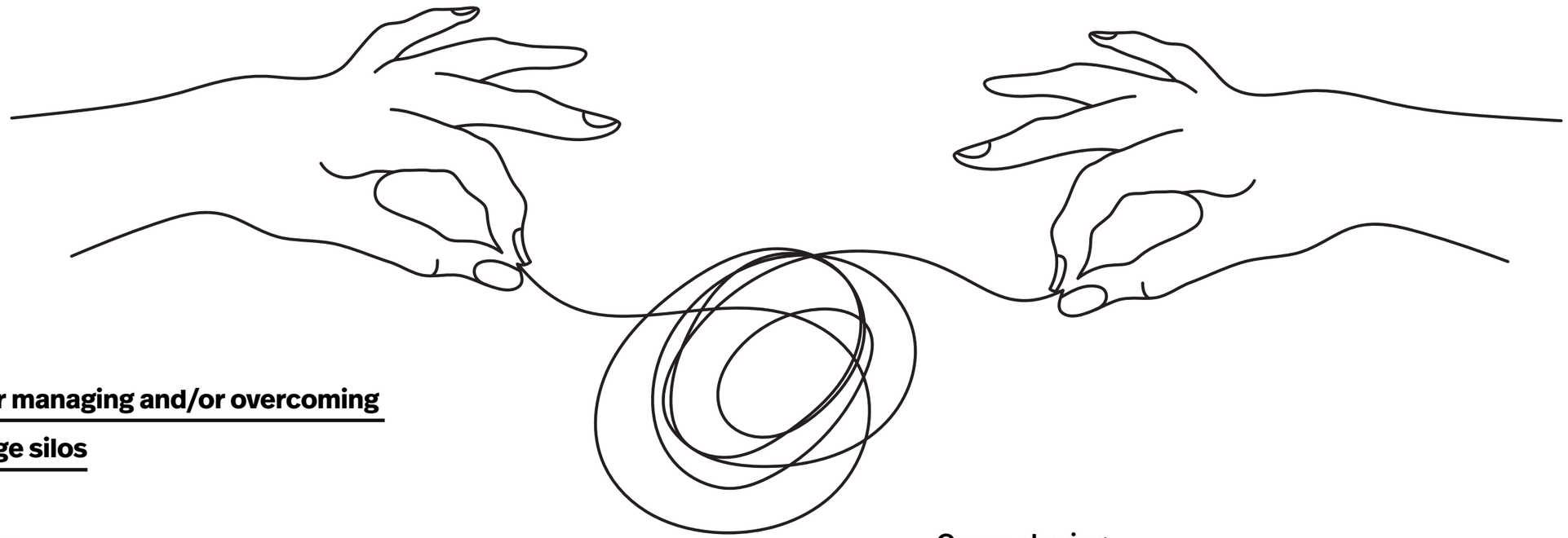
Here's the problem with knowledge silos: too much unique knowledge in one person's brain. If that person doesn't share what they know or isn't available, it could jeopardize the entire team. Not to mention the potential business risk and expense.

But you can't always un-silo your way to the ideal team environment. In fact, having a few silos can be a good thing – as long as the team knows what's in each one and how to access the information.

A quarterback may have a unique command of the playbook, but there is always a backup ready on the sidelines.

Types of knowledge silos

- **Technical skills**, like being the only one on the team to know Java or Python
- **Ancillary knowledge in some field**, such as a background in statistics
- **Historical knowledge**, like the seasoned vet who's been around long enough to know the whole code base
- **Institutional knowledge**, like being privy to other eras in the company's history, or how the company's products came to be.



Hacks for managing and/or overcoming knowledge silos

Awareness

It starts with knowing who possesses what deep skills or background.

Collaboration

Think creatively about pair programming or how you assign pull requests.

Self-assessment

Have each team member ask: What are my silos? What can I share?

Goal setting

Consider adding knowledge sharing to your `1:1`s or performance reviews.

Group sharing

Schedule brown-bags to let devs present their unique knowledge to the team.

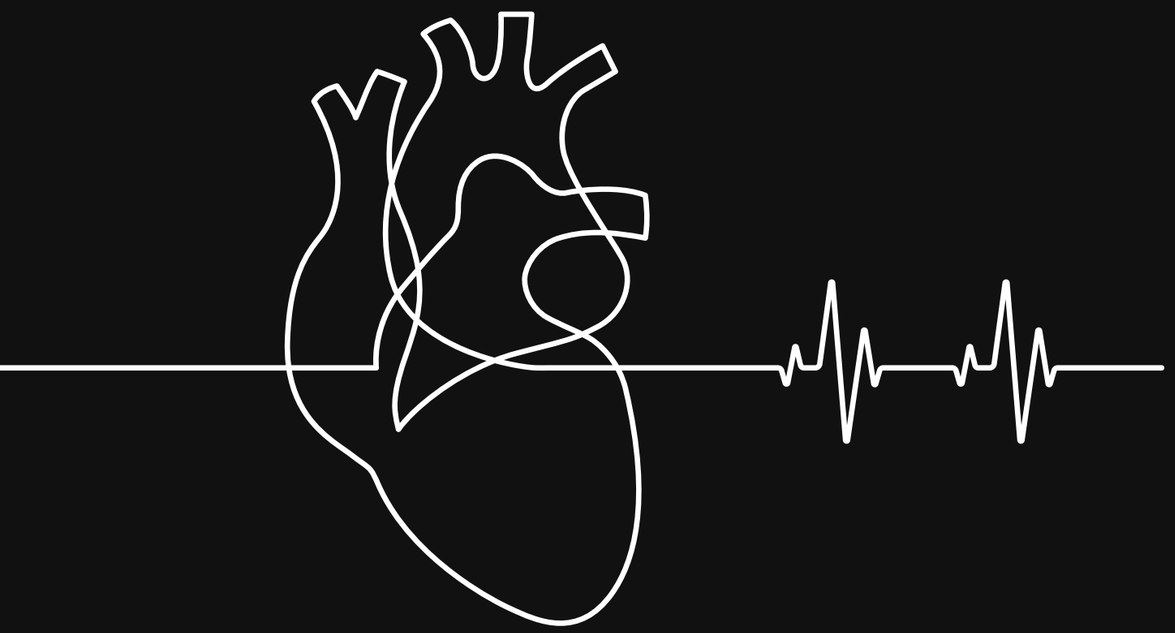
Documentation

Documenting code isn't sexy work, but it's a way to share what you know.

Visualization

Use management tools that help the team visualize where silos persist.

The human side of team building



Sometimes, building a better team dynamic is as basic as examining the different personality types that make up your dev team. Fast Company broke out a typical work team into four key role types, all of which are essential to team success:

1. The **BRAIN** drives others to take action in pursuit of a goal.
2. The **VOICE** gets the big picture and motivates the team.
3. The **SHARP EYE** focuses on details to get things done the right way.
4. The **HEART** promotes community, consensus, and collaboration.

Who plays each role on your dev team? Which one are you? Are any of the four types missing? And what can be done to better nurture these personality types?

With the right role balance, you can build a harmonious team while playing to the strengths of each individual.

Fostering empathy, building trust

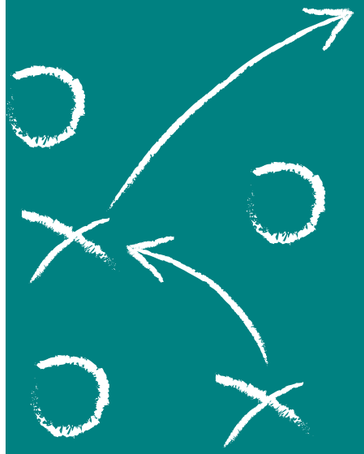
When you break down the specific dynamics that enable healthy teams, empathy and trust top the list.

On a practical level, teams function best when everyone produces. But it's just as practical to consider and feel empathy for one another's perspectives and roles. For managers and execs, this means balancing your drive to win with real human connection.

What makes your teammates unique? How are you alike? What's at stake for each role? Push yourself to ask thoughtful questions that put you in their shoes.

Devs also want to be trusted with applying their knowledge and managing workloads. At the same time, it's important for devs to trust their managers, and for managers to trust their execs. When empathy and trust are shown across all roles, the whole team wins.

Related reading: A manager's guide to effective engineering



Builds Trust

- Supporting developers at their unique levels
- Nurturing a culture of healthy sprints and personal well-being
- Using data to guide improvement
- Clearing roadblocks to productivity
- Granting autonomy to individual contributors
- Sharing data across the org

Erodes Trust

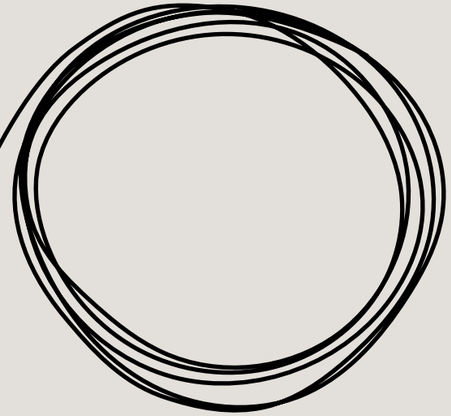
- Stack-ranking developers by lines of code
- Rallying around ambitious deadlines at the expense of burnout
- Using data to point out flaws
- Enforcing a productivity quota
- Managing with a top-down perspective
- Limiting data to leadership

Managing stress

The World Health Organization (WHO) defines burnout as “a syndrome conceptualized as resulting from chronic workplace stress that has not been successfully managed.”

Burnout isn’t just a personal problem – it’s an organizational problem. For the individual, burnout can produce exhaustion, cynicism, and a loss of efficacy. For the company, that translates into lower productivity and a potential impact on cycle time.

Teams should be especially mindful of the potential added stress and isolation some feel from ongoing remote work. Check in with your teammates – and yourself – to make sure no one feels detached.



Questions to ask when checking in

- Are you balancing your workload effectively?
- Are multitasking and context switching diluting your focus?
- Are you making time for Deep Work?
- Are you getting too many mid-sprint work or scope changes?

What to do when you feel stressed

- Check in with your manager and teammates.
- Pull data to support the problem, such as where your time goes.
- Review current priorities – yours AND the team’s.
- Prioritize tasks that energize you over those that deplete you.
- Talk to someone you trust – like a friend, a therapist, or your dog.

Keeping burnout at bay

Burnout is a high-impact obstacle. It takes way more time and energy to recover from burnout than it does to prevent it.

Consider some of these stress-busting burnout-blockers:

Self-assess regularly.

Get some distance. It may sound self-helpy, but make time to step away from your stress. What really energizes you? What depletes you? What do you need to feel seen, heard, recognized, and effective?

Take time off – for real.

Detach. Take genuine breaks. Embrace "Do Not Disturb" mode. Plan meaningful mini breaks throughout your day, and when you go on vacation, truly "vacate" your job.

Relax your brain.

Make time during those breaks to embrace lighter activities. Play games. Cook a meal. Challenge yourself, if you must, but not so much that it adds to the stress.

Balance your mastery.

Spend time on passions that have nothing to do with work. You know what they are – now go do them and restore your equilibrium.

Related reading: A dev's guide to burnout

How to ask for help

- When you start to feel stressed, take a deep breath or a short walk. And if that doesn't help, try reaching out to someone – a teammate, friend on another team, or your manager.
- Be truthful about your work situation. Too many epics? One mismanaged epic? Lack of purpose? Be specific and share facts. If the problem is a lack of clarity on facts, share that too. It's OK if you don't know what's needed to fix the problem.
- Keep your communication channels open. Asking for help is much easier when there's an established place to do so, such as a dedicated Slack channel or informal DM check-ins with your manager. Make the conversation ongoing so burnout doesn't take you by surprise.

> "When you're STRUGGLING as a dev, it can be really easy to just not say anything to anyone and hope everything works out well. It takes having real TRUST on a team to be open and say: I need a little HELP."
>
> **–Brian Park**
> Dev Manager at Uplevel

Deep Work and the art of blocking distractions

We surveyed over 3,000 engineers for our Uplevel State of Engineering Report. A concerning 50% of respondents reported “not enough time to do work” on a regular basis (2+ times per week) – 20% feel that daily.

Deep Work, a period of two or more hours without interruptions, is a cornerstone of engineering effectiveness. Deep Work means freedom from ongoing context switching between this Slack thread, that meeting, this unanswered question.

Deep Work is important for maintaining productivity and staving off burnout. So, switch up the game. If you’re spending all day playing Whac-a-mole, set aside time in your daily or weekly calendars for Deep Work. Managers and devs alike can benefit from this shift.

The strong link between Deep Work and burnout

Uplevel examined the relationship between Deep Work and burnout risk by analyzing existing customer teams and exploring how Deep Work was impacted by always-on culture.

We used an “Always On” Uplevel metric that takes stock of burnout risk behaviors such as long work hours and responding to Slack messages on weekends. The metric accounts for each person’s individual schedule, which can vary day-to-day – especially with the blurring of boundaries between work and home.

The results were clear: Teams with more Deep Work tend to have lower Always On scores. In other words, when engineers have sufficient time to focus, they require less overtime.

Balancing player focus and team engagement

The best players in any sport are the ones who keep their heads in the game while maintaining awareness of the team around them.

This balance is true on dev teams as well. On the one hand, you play a unique role in helping your team get work done. At the same time, you need to be a team player, staying connected and keeping up with overall progress.

It’s vital to maintain a healthy balance of engagement versus focus, collaboration versus Deep Work. Dev teams rely on this balance. It helps keep each team member connected to the larger picture while giving true context to their efforts and value. At the same time, it allows leadership to focus on the “big picture” and prevent burnout.

Tips for keeping remote teams engaged

Online jam sessions

Try a digital whiteboard like Miro or Mural so team members can contribute wherever they are, getting you closer to a classic brainstorming session.

TGIF

Try “Do Not Disturb (DND)” Fridays, where everyone shuts off notifications and gets a final stretch of uninterrupted Deep Work before closing out for the weekend. You could also consider DND Monday mornings, or a monthly mental health day.

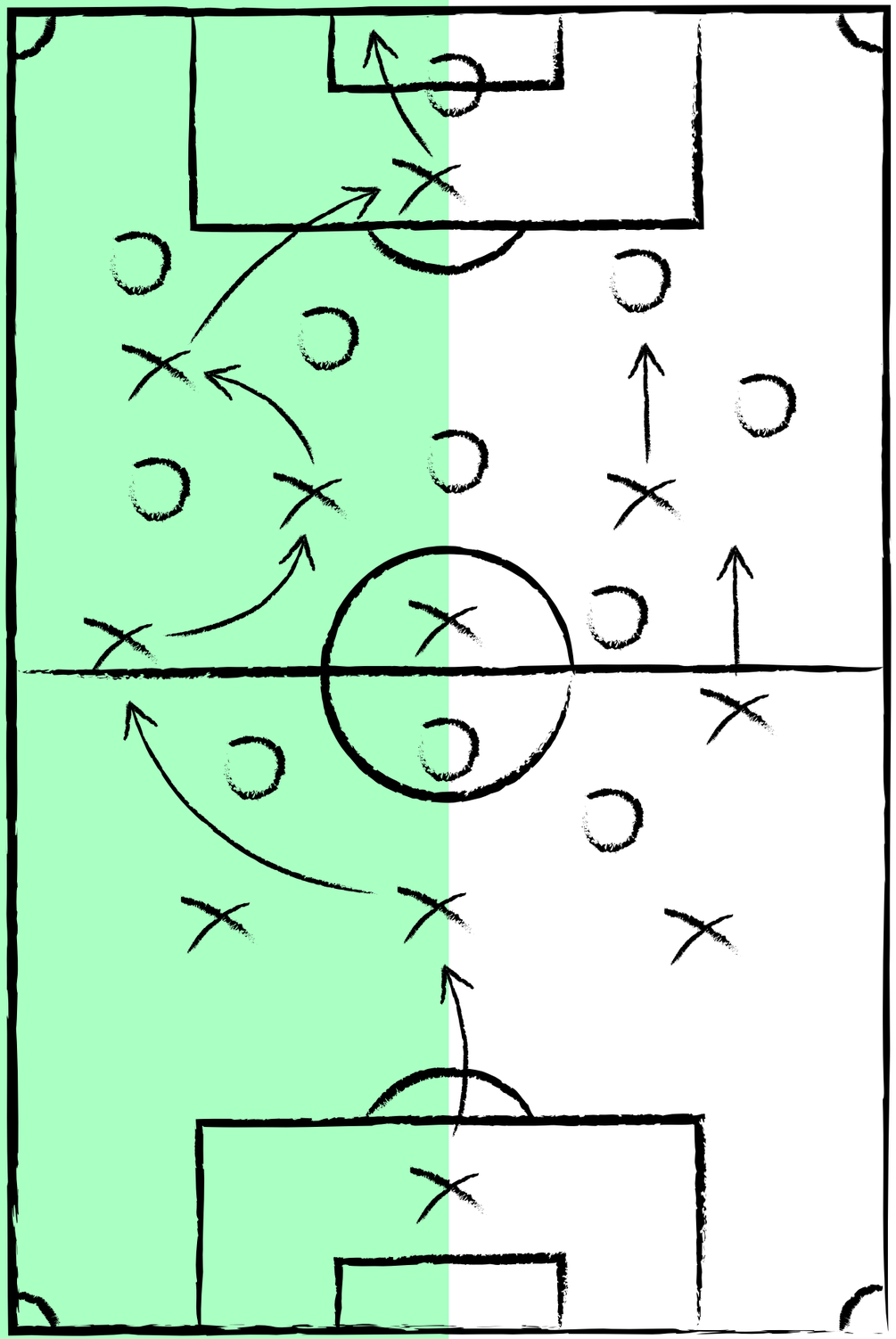
Vibe checks

Make space for team feedback. Dedicate office hours to work culture, send weekly pulse surveys, or find other ways to collect your team’s thoughts.

1:1s 2.0

Rethink the structure and purpose of your 1:1s. Arrive with clear data so you can get through the work portion of your check-in efficiently. Then spend the rest of the time focused on personal well-being.

Strategic play calling for better 1:1s & sprint retros



In sports, teams watch game film to identify successes and areas for improvement. Dev teams use **1:1s** and **post-sprint retros**.

Dev 1:1s are often downsized to quick coffee runs or casual chats on the way to lunch. In the age of social distancing, they may consist of a few bullet points in a Slack message – or not happen at all.

Sprint retros are a routine part of the agile process. But after weeks of coding and accrued sleep debt, devs often struggle to remember specific issues.

`

Let's wrap up our playbook with some tactical suggestions for making the most of these key dev team and project elements.

Improving 1:1s

Done well, weekly 1:1 meetings provide a forum for career planning, roadblock removal, and big-picture thinking. Here are some tips for more effective 1:1s:

Hold your 1:1s regularly.

Instead of catching up whenever you find time, set up a recurring meeting with each team member. This gives the meeting a greater sense of importance and rhythm.

Change up the location.

We spend enough time in conference rooms. Meet at the cafe next door or have a walking meeting. If working remotely, just make sure you have a reliable connection wherever you choose to work.

Share an agenda.

Work together in advance of the meeting to include topics you both want to discuss.

Keep it gossip-free.

As satisfying as it may feel to vent, the highest priority of your 1:1 is to safely communicate problems and needs.

Discuss team and career aspirations.

Shape the conversation around high-level concepts like improving the way teams collaborate or working toward a promotion.

Take notes.

Start an evergreen Google Doc for each direct report, returning every week to add notes, track patterns, and save ideas for later. This can help you identify progress (or issues) and makes life easy when it comes to annual reviews. Just make sure to save it securely.

Sample questions for effective 1:1s

- What was the highlight of your past week?
- What were your top accomplishments?
- How's your well-being (or stress level) right now?
- Where did you get stuck?
- Looking ahead, what is blocking you from doing your best work?
- Is there anything I can do better to improve your experience here?
- Is there anything I can do better to help develop your career?
- What other, if any, feedback do you have for me?

Pro tip: To make 1:1s truly actionable, you need data from sources like Jira activity, PRs, Deep Work time, and burnout risk. Take a look at this data in action.

Improving sprint retros

The sprint had flaws, but can you remember them? And how can they be remedied? Here are some tips for improving sprint retros:

Gather feedback throughout the sprint.

Don't wait until the sprint retro. Think of it this way: You may remember what you had for breakfast this morning, but what about last Tuesday? Teams can collect much more acute feedback in real time. Prompt devs for a thumbs up or down on specific Jira tickets, call out red flags, and provide a comment space for questions and ideas.

Review items added mid-sprint.

Were they cross-dependent issues that could only be added once others were complete? Breaking news from leadership? What could have been done to better anticipate the additions?

Understand setbacks as symptoms, not problems.

Zoom out to isolate the problems and consider the long-term view.

Review your project health AND your people health.

Did the team complete the sprint at the expense of high burnout? Did people work long hours and on weekends? Were they context switching at unsustainable rates?

Pro tip: Just as with 1:1s, data matters. Read how Uplevel uses data to improve sprint retros.

Uplevel is your engineering insights solution that leverages data from everyday developer tools to help you reliably meet your sprint goals without burning out your team.

Schedule a demo today.

uplevelteam.com

Uplevel