SMOLDER ·FRAGILE

Smoke Signals

Recognizing and Addressing Hidden Technical Risk

> *The most dangerous tech debt is the kind that doesn't show up in dashboards.*

Your systems are running and your customers aren't revolting. Your leadership team looks at uptime numbers and sees green, but somewhere in your engineering organization, a few people are holding their breath every time something changes, because they know how fragile the thing underneath actually is.

If that sounds familiar, you're in a smolder.

A smolder is the most common state for engineering organizations that have been operating for more than a few years. It is not a crisis yet. It's the gap between what leadership believes about the health of the systems and what the people closest to the code actually know. It persists because, from the outside, everything looks fine.

> **This paper is about learning to see the smoke.**

Why Smolders Are the Most Common State

Every engineering organization accumulates debt. That's not a moral failing. You made reasonable decisions under real constraints. Ship the feature. Close the deal. Hit the date. Those decisions added up, and some of them left residue.

The residue itself is manageable. What makes it dangerous is that it becomes invisible over time. The workaround that was supposed to be temporary becomes the way things work. The deployment process that requires three people to coordinate becomes "just how we release." The system that one person understands becomes everyone else's black box.

Smolders persist because they're comfortable. Not pleasant, but comfortable. The team has adapted. They've built muscle memory around the fragility. They know which systems not to touch, which deploys need extra care, which person to call when something breaks. It works. Until it doesn't.

The Visibility Gap

Leadership typically sees uptime metrics, sprint velocity, feature delivery timelines, and customer satisfaction scores. Those numbers can look healthy for a long time while the foundation deteriorates underneath.

The team typically knows which parts of the codebase nobody wants to touch, which tests don't actually catch anything, which "quick fix" from two years ago is load-bearing, and how many things have to go right for a release to go smoothly.

Nobody is lying. The structure just rewards not surfacing problems. That means fixing the structure is on you as the leader. Without deliberate systems for checking in on what's happening on the ground, systems that don't feel like micromanagement, leaders are always going to get a version of the truth that makes things look best. Nobody wants to fail their boss. Nobody wants to be the one who says "this is broken and I don't have a fix." So they adapt. They work around the problems and leadership never hears about it.

The gap gets worse when leaders require the people who report a problem to also deliver the solution. On the surface, this sounds like good delegation, empowering your team to own their challenges. In practice, it punishes honesty. If raising a concern means you just assigned yourself a project on top of your existing work, people learn quickly to stop raising concerns.

Solutions should be a collaborative effort. Your direct reports see the problem from the ground. You have institutional power, budget authority, and cross-team visibility they don't. If you put it on your people to both identify and solve every problem, what exactly is your role as a leader? Paving the way for a solution is your job. Collaborating on what that solution looks like is the work you should be doing together.

There's another pattern that's equally damaging. Leaders assume that their experience gives them a more accurate picture than their team's daily reality does. "They're just complaining. They don't know how hard it is. I've seen worse." Maybe you have, but you're also far removed from the ground. You don't know what you don't know. A VP named Pat once discovered, months after the fact, that their QA team had been carrying nearly double the workload everyone assumed. The additional responsibilities had accumulated gradually, never surfaced in any planning conversation, and nobody had thought to ask. When you're that far from the daily work, invisible burdens stay invisible unless you actively look for them.

> **The question every leader in a smolder should be asking their team is not "how's it going?" It's *"what am I not seeing?"* Then believing the answer without requiring a presentation to prove it.**

Knowledge Concentration: The Slow-Motion Risk

Every engineering team has people who know more than others. That's natural. Is that knowledge accessible or locked inside individual heads, or inside direct messages between two people who may not be there next quarter?

A key person leaves. They resigned or they were part of a reduction. Either way, the immediate aftermath looks fine. They did good work while they were there. Systems are stable, documentation exists for the obvious things, and the team adjusts.

Then a month later, something breaks. Not catastrophically, just enough to require deeper knowledge than anyone remaining has. Someone digs into a system the departed person maintained and discovers it's more complex than anyone realized, or a process that seemed automatic turns out to have been manually managed by someone who's no longer answering Slack messages.

By the time these fractures surface, the organization has forgotten that the departure is the reason. Leadership sees a production issue and blames the current team. "Why didn't anyone catch this?" The source of the problem was the knowledge that walked out the door weeks ago.

This compounds when knowledge lives in private channels. If two engineers have been troubleshooting a critical system via DMs for a year, that's a year of institutional knowledge stored in a format nobody else can access. Let both of those engineers go in the same quarter, which happens more often than anyone admits, and that knowledge is effectively destroyed. Nobody is going to excavate thousands of direct messages to reconstruct what those two people understood.

The fix is building practices where knowledge is shared in public by default. Decisions made in shared channels. Troubleshooting done in team-visible spaces. Architecture context recorded where the next person can find it. Not because you're planning for people to leave, but

because knowledge that's only accessible to one person is not really organizational knowledge at all. This won't happen in an organization where people hoard knowledge as job security. Establishing that climate of trust is a leadership responsibility.

Case pattern: The Carol Problem

We worked with an organization where standing up a new server was a three-month process. That alone is a red flag, but the real problem was in the details. When we mapped the process during discovery with every person involved, the workflow looked like this: Department A hands off to Department B, who hands off to Department C, who hands off to Carol, who hands off to Department D, who hands off to Department E.

Carol. One person in the middle of a multi-department, business-critical process. Ten manual steps that only she knew how to perform. If Carol went on vacation, the entire process paused until she got back. If Carol left the company, they would not be able to stand up new servers. They would not be able to serve new customers. The business would functionally collapse, and nobody had flagged this as a risk because the process was "working."

That was not the only problem hiding in the same workflow. The person responsible for decommissioning the old server after migration, deleting the data that had been copied to the new one, had been let go at some point. Nobody reassigned that step, so data was not being cleaned up. The company could not figure out why they were burning through server capacity so fast. The answer was sitting in a process that nobody had looked at end to end since it was first designed.

Two different forms of knowledge concentration and two different kinds of debt accruing simultaneously, all inside one process. Until someone sat down and mapped the whole thing with every person in the room, none of it was visible.

> **That's a smolder.**

Mapping Your Smolder

You don't need a formal audit to understand where your smolder lives. You need candid answers to a few questions, and you need to ask the right people.

Start with your engineers, not your dashboards.

Ask them: *What's the thing you're most worried about that nobody's working on?* Ask them: *If you left tomorrow, what would break that nobody else could fix?* Ask them: *What do you work around instead of fixing?*

These questions will surface the real risk map faster than any scanning tool or consultant report.

There's a prerequisite that makes all of this work: psychological safety. These questions only produce candid answers if the people you're asking believe that candor won't be punished. If the last person who flagged a problem got assigned to fix it on their own time, or got told they were being negative, or watched their concern get dismissed in a leadership meeting, your team has already learned what happens when they speak up. They will tell you things are fine. They will give you the version of the truth that's safest for them, and you will continue to not see the smoke.

Before you ask these questions, you have to have built the kind of environment where the answers can be real. If you haven't, that's the first thing to fix, because every other step in this section depends on it.

Then categorize what you find by consequence, not severity.

Not every piece of tech debt matters equally. The right question is not "how bad is this code?" but "what happens to our customers and our team if this fails?" A messy but isolated component is different from a messy component that sits in the critical path of your most important workflow.

Map what you find into three buckets:

- **Customer-facing risk.** Things that, if they fail, your customers experience the failure directly. Slow pages, broken workflows, data inconsistencies they can see.
- **Team-facing risk.** Things that make your engineers slower, less confident, or more error-prone. Flaky tests, manual deployment steps, undocumented organizational knowledge.
- **Compounding risk.** Things that make other problems worse. Knowledge concentration, lack of monitoring, no incident review process.

This categorization matters because it changes how you make the case for fixing things.

The Permission to Fix

This is where most smolders stall. The team knows what's wrong. They've known for a while, but they can't get time or budget allocated because the problems aren't visibly on fire yet.

The usual mistake is framing tech debt in technical terms. "We need to refactor the payment service." "Our test coverage is too low." "The CI pipeline is slow." These are accurate descriptions of the problem, but they don't connect to anything a non-technical executive cares about.

The translation that works is impact, specifically impact on customers and on the people doing the work.

Frame a database migration not as "modernizing our data layer" but as "right now, when a customer's data request takes 45 seconds instead of 2, that's this system, and it's going to get worse as we add customers."

Frame a CI pipeline overhaul not as "improving developer experience" but as "our engineers currently wait 40 minutes to find out if their change broke something. That means every bug

fix takes a minimum of half a day in a cycle of change, wait, discover, fix, wait again. We're paying senior engineering salaries for people to watch progress bars."

Frame infrastructure maintenance not as "keeping the lights on" but as "the last three customer escalations all traced back to the same system we've been saying we need to address. Each escalation costs us [X] hours of engineering time and [Y] hours of customer success time. The maintenance work costs less than three more escalations."

The pattern is the same every time. Connect the boring technical work to something the business already measures and cares about: customer experience, employee productivity, escalation costs, revenue risk, or whatever your leadership cares about most.

There's one more framing that's particularly effective for smolders: *insurance.* You're not asking to fix something that's broken. You're asking to prevent something from breaking during the worst possible moment, when you land that big customer, when you're trying to hire, when a competitor is pressuring you to move faster. The organizations that handle those moments well are the ones that cleared the brush before fire season.

Building Burn-Down Momentum

You don't fix a smolder with a big-bang initiative. Those get funded, started, and abandoned when the next business priority shows up. You fix a smolder with a sequence of small, visible wins that build political capital for bigger investments.

Start with something the team feels every day.

Pick the thing that causes the most daily friction, not the biggest risk and not the most important system. A flaky test suite that makes people distrust CI. A deployment process that requires a specific person to be available. A monitoring gap that means the team finds out about problems from customers instead of alerts.

Fix that first. Fix it visibly. Make sure the people who approved the time to fix it see the result. It builds credibility with the team whose daily pain you've eliminated, and with leadership that you know what to fix and the benefit it provides.

Then connect the win to the next problem.

"Remember how we fixed the deployment bottleneck last month? That same pattern, one person holding critical knowledge of a manual process, exists in three other places. Here's the list. Here's what we'd fix next and what it would save."

Each win makes the next ask easier. Each fix demonstrates that the team can identify problems, propose solutions, execute, and deliver measurable results. That track record is what gets you from "fix the flaky tests" to "rebuild the data pipeline."

Protect the time.

The biggest threat to burn-down momentum is context switching. A team that has 20% of their time allocated to debt reduction but gets pulled into urgent feature work every sprint effectively has 0% allocated. If you're serious about addressing the smolder, the time has to be protected by having someone with authority defend it when the pressure comes.

What a Smolder Does to the Team

In a brush fire state, the team is functional and the human cost is invisible. In a smolder, the cost becomes structural. The team is still working together, still shipping, still solving problems, but the way they work together has changed in ways that are hard to see unless you're looking.

The most damaging shift is that people stop trusting that raising problems will lead to change. They've flagged issues before. They've explained the risks. Nothing happened, or worse, they got assigned to fix it on top of their existing workload, so they adapt. They stop raising concerns. They stop advocating for improvements. They focus on their own work and accept the problems they can't change.

This looks like a team that's functioning. It feels, from the inside, like a team that's given up. Not on the work itself, but on the idea that the organization will invest in making the work better. The individual contributors are still competent and still care about their craft, but they've stopped believing that the team, collectively, can move the needle on the systemic problems. They've stopped collaborating on the hard stuff and started just getting through the day.

The second shift is more subtle. In a healthy team, people lean on each other. They ask for help. They share context. They pull each other into problems because they trust that the help will be reciprocated. In a smolder, people start insulating. They build expertise in their own area and don't invest in understanding anyone else's, to protect themselves. If the organization isn't going to address the systemic problems, the safest move is to make yourself indispensable in your corner and keep your head down.

This is the knowledge concentration problem from the other direction. People are actively choosing not to share, because sharing means taking on more responsibility in an environment that's already stretched thin.

The third shift is what happens to the team's relationship with leadership. In a brush fire state, there's still goodwill. The team believes leadership will listen if something important comes up. In a smolder, that goodwill has eroded. The team has learned through experience that leadership either doesn't see the problems, doesn't prioritize them, or doesn't follow through on promises to address them. Communication becomes performative. Status updates are crafted to say the right things rather than the true things. The gap between what the team reports and what the team knows grows wider.

Rebuilding from this state requires more than fixing the technical problems. It requires rebuilding the team's belief that raising a problem is worth the effort, that collaboration on systemic issues is valued, and that leadership will follow through. Technical fixes without that foundation will be undermined by a team that no longer trusts the process.

When a Smolder Becomes Something Worse

Smolders are stable, and that's both their defining characteristic and their danger. Stability is not permanent. Certain events can tip a smolder into a crown fire almost overnight.

A key person leaves and the knowledge gap surfaces a month later as cascading failures. A big customer arrives and the platform can't handle the scale or the customization without duct tape. A competitor moves faster and the pressure to ship outpaces the team's ability to ship safely. A security or compliance event reveals that the systems everyone assumed "worked fine" don't actually meet the standards anyone thought they met.

The organizations that weather these moments are the ones that saw the smoke and acted before the wind changed. The ones that didn't are the ones calling for help at 2 AM.

If you took the quiz and landed here, you're in a position to act before the crisis. That's genuinely good news. Most organizations don't get a clear signal that something needs attention while they still have time to address it calmly.

Now what

You have a read on how serious this is. The next step is turning it into a plan for the one thing most in your way. As you read, mark this up: what resonated, what didn't fit, and what you're still unsure about. Then write down the three questions you still have.

Bring your question to TPM office hours →

Not ready for that? Ask us a question for free. We answer the ones that help the most people, on the blog and on socials.

About Understory Collaborative

Understory Collaborative is a team of seasoned technologists who specialize in the work that happens beneath the surface. We're smokejumpers. We help organizations clear the brush before a smolder becomes a bigger fire, and we bring the experience and clarity to help you see what's actually happening on the ground.

Whether you need a focused assessment, hands-on technical leadership, or a partner to help you build the practices that prevent the next crisis, we meet you where you are.

If anything in this paper sounded familiar, we should talk. Reach out at contact@understorycollab.com.