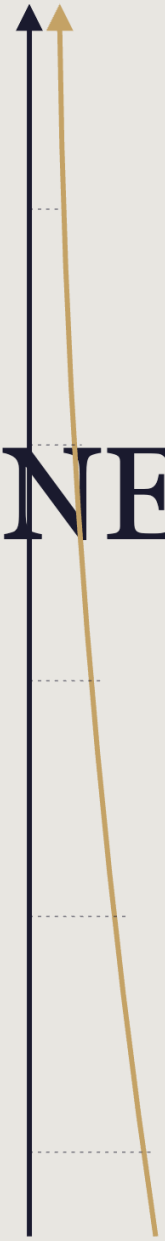
BRIDGING THE GAP BETWEEN WHAT ORGANISATIONS SAY THEY DO AND WHAT THEY ACTUALLY DO

LEADING AGILE WHEN NO ONE AGREES



READING THE DRIFT
IN COMPLEX SYSTEMS AND MATRIX ORGANISATIONS

KYLE HAUSLAIB

A note on the scenes in this book

The scenes in this book are composites. Names, roles, timings, locations, organisational structures, technical domains, and the specific details of incidents have been altered, blended across multiple settings, or fictionalised entirely. Several scenes draw on situations that occurred in different industries, different countries, or both, and have been reset into a single illustrative context for the sake of the argument. Where a scene reads as if it took place at a specific organisation I have worked for, it almost certainly did not, in the form described. The patterns are real enough. The people in the scenes are not, at least not in the form they appear here.

The book also describes leadership behaviours, organisational decisions, and management practices that may resemble situations the reader has encountered. That resemblance reflects how widely these patterns occur, not how identifiable any particular instance is. Nothing in this book is intended as a description of any specific colleague, employer, or incident, past or present.

Where I describe my own mistakes, I have tried to be accurate to the texture of the experience while generalising the specifics. The mistakes are mine. The details around them have been adjusted.

This is not legal, employment, medical, financial, or psychological advice. It is a practitioner's account of the gap between what organisations say they do and what they do, written for other practitioners working in the same gap.

Foreword: Why I Wrote This Book

This isn't a book that explains agile. Plenty of those exist already, and most of them handle the mechanics better than I would. The frameworks, the ceremonies, the role definitions, all of it gets covered with the kind of patience the basics deserve. If that's what you came for, the bookshelf has options.

I wrote this because of what tends to happen after those books. Training is done. The transformation kickoff is finished. The framework is implemented, the roles assigned, the boards up on the walls. Everything looks right and somehow almost nothing feels right. Ceremonies run on time and the conversations inside them are hollow. Boards get updated daily and what's on them has very little to do with what is happening in the code, in the teams, or in the meetings where the real decisions get made.

That distance is what this book is about. I have a name for it in these pages. The Drift.

What the Drift is

In most organisations there are two systems running at the same time, even if no one says so. The first is the Stated System, which is what the organisation says it does: the org chart, the written process, the planning cadence, the reported metrics, the prescribed roles, the

language used in steering committees. The version of the organisation you can describe in a slide.

The second is the Lived System, which is what the organisation actually does. Where decisions get made. How priorities get set. Who carries which piece of knowledge. Whose calendar sets the throughput of which work. Which channels carry honest signal and which forums punish it. The version of the organisation you can only describe after you've worked inside it long enough to see it.

The Drift is the distance between the two. Some Drift is normal. Organisations adapt to context faster than their documentation updates, and that's healthy. What is not healthy, and what this book is mostly about, is Drift that widens without anyone noticing. There is usually a day when the Stated System stops being a useful description of the organisation and becomes a fiction that's maintained for reporting purposes, and almost no one is watching for that day.

Where this book comes from

I started taking notes for what became this book without intending to. It accumulated. A standup that felt empty. A plan that didn't survive contact with the first sprint. A planning board that I remember being proud of and that was obsolete inside two weeks. Each of these felt like its own small problem at the time, the kind of thing you put down to a bad week or a tired team or a difficult personality. Across years and across teams, what I started to see was something less like failure and more like adaptation. The Stated System held still while the Lived System moved around it, continuously, until the picture on paper was clean and the picture in the work was something else entirely.

I have worked across most of the agile leadership roles, mostly inside large engineering organisations. Places where the small workplace rituals are closer to a unifying force than any retrospective ever managed to be, where the seasonal shutdown weeks reset every Kanban board whether anyone planned for it or not. Before that I held a smaller workshop role in a different country and a different industry, in a setting that taught me most of what I now know about how informal authority operates. I started in software development. After that came Scrum Master work, where I struggled for a long time to run retrospectives that produced honest conversation. After that, Release Train Engineering, where I watched carefully built increments come apart inside the first sprint. These days I do Solution Train work, holding several Agile Release Trains in alignment while the informal networks around them quietly operate on a different set of priorities. Every role has changed the angle. The Drift hasn't.

I should also be honest that some of what's in this book is here because I got it wrong. There is a chapter near the end about being the bottleneck. I am the person it is describing. For something close to two years, every escalation, every awkward stakeholder conversation, every ambiguous decision in two of the trains I worked with came to me before it went anywhere else. The teams got slower at deciding. I got better at deciding. I told myself this was leadership. It was not. I had made the system depend on me, and the version of me that did that wasn't who I wanted to be. Most of the patterns in this book I have either lived inside or contributed to, often both.

What Reading the Drift means

The leadership practice this book is trying to teach is not closing the Drift. Closing it isn't really possible, and it's worth being honest about

that early. The Stated System and the Lived System exist for different reasons. The Stated System makes coordination at scale possible: budgets, headcount planning, regulatory reporting, customer commitments. The Lived System makes work at the level of an actual day possible. It runs faster than the planning cycle, it adapts more quickly than the backlog, and it tends to be more honest than the dashboard. Both serve real needs. Asking the Drift to disappear is asking one of those systems to do the other one's job.

Reading the Drift is something else. It means seeing the gap clearly enough to lead inside it. Knowing which system is more real for which decision. Knowing where the Drift is widening, and where it isn't, and which conversations have started to take place in the wrong system for the work being done.

Once you can read the Drift, three questions tend to structure most of the work this book describes. What does the Stated System say should happen here? What does the Lived System actually do here? Who pays for the gap, and what would close it? The first question has an answer in writing somewhere. The second only has an answer in the heads of the people who do the work. The third is where leadership starts, because the cost of the Drift is rarely paid by the people who designed the Stated System.

Who this book is for

This book is for people who are past the framework stage. Who have felt the Drift in their own work, even if they didn't have a word for it, and who are now responsible for outcomes inside imperfect systems. Release Train Engineers who spend more time on organisational politics than on flow. Scrum Masters who know their retrospectives are not changing anything and aren't sure why. Product Owners

caught between stakeholder demands and team capacity. Engineering managers trying to hold delivery pressure and team health in the same hand. Leaders at any level who have stopped asking whether they are doing agile right and started asking why it isn't working the way they expected.

It won't give you clean answers. Those don't really exist in complex systems. What it offers is a way of seeing the forces shaping your organisation, the patterns underneath the behaviour you're watching, and some of the moves available to you when you don't have formal authority over most of what affects your outcomes.

How this book is organised

The book is in four parts.

Part I, *The Stated System and What Hides Inside It*, walks through the patterns that let organisations keep up the appearance of agile while the Drift widens underneath. Each chapter takes one pattern, from empty standups through to disappearing improvement, and tries to explain why the pattern shows up and what it costs.

Part II, *Reading the Drift*, steps back from individual patterns and looks at the system-level forces behind them. The two parallel systems. The gravitational pull toward control. The way metrics turn into culture. The paradox that makes change programmes resist change.

Part III, *Leading Inside the Drift*, turns from observation to action. How to lead when you don't control most of what affects your outcomes. How to work with stakeholders who keep bypassing your systems. How to make planning useful when reality starts eroding the plan in the first sprint. How to lead across cultures and time zones without

defaulting to the dynamics of whichever group is dominant in the room.

Part IV, *What Closes the Drift*, brings it together. What a functioning system looks like in practice. A practical toolkit of diagnostics and conversation starters. A closing reflection on the quiet, ongoing contract that leadership turns out to be.

Organisations rarely reject agile outright. They reshape it until it fits the structures already in place, and in doing so they remove the conditions that made it valuable to begin with. The work is to see that process clearly while it is happening, and then to decide what to do with what you've seen.

Worksheets, scenarios, and retrospective formats referenced throughout the book are available as a free companion at kylehauslaib.com/companion. The full set lives as editable Markdown on GitHub for anyone who wants to fork or translate it.

CHAPTER 1

The Standup That Says Nothing

Standups don't usually announce that they have stopped working. The rhythm gives it away if you're paying attention to the rhythm and not just to the updates. People speak in roughly equal lengths, in roughly the right order, with brief eye contact and almost no follow-up questions. The Scrum Master nods at the end of each update. The board, on the wall or on the screen, gets updated as people speak. The meeting ends on time. Nothing in it produces information that anyone is going to use.

It took me weeks to understand what I was watching, the first time I saw this in a team I was working with. The team was pleasant. The standup was punctual. The board was tidy. By every visible measure, things were fine. My early diagnosis was that the team needed coaching on what good updates looked like, so I gave them some. The standup got slightly more polished. None of the underlying problems surfaced any earlier than they had before, which is to say, they surfaced when they could no longer be hidden.

What I was looking at, I eventually realised, was a meeting that had quietly reorganised itself around a different goal from the one its name suggested. The standup wasn't a coordination ceremony anymore. It had become a status broadcast: people reporting in to

confirm they were still doing what they had said they would be doing. That's a useful function for somebody. Not for the team.

There was a developer on one of the trains I worked with at the time. I'll call her Mei, although that wasn't her name and the patterns I'm describing weren't all from one person. She was a careful, generous engineer, the kind who quietly fixes things other people walk past. She usually went third or fourth in the standup. Her updates were brief, clear, entirely procedural: what she'd worked on yesterday, what she was working on today, whether anything was blocking her, which was almost always no.

One morning, after a particularly empty round of updates, I asked her on the way out of the room whether she had everything she needed. She said yes, in the same voice she'd just used in the meeting. We walked in silence for a few metres. Then she added, almost as a side comment, "I had a question about the schema change yesterday. I was going to bring it up. But Marcus mentioned the milestone in his update, and I could see where the meeting was going. I'll catch up with the architect later."

I asked her what would have happened if she had brought it up.

She thought about it. Then: "Probably nothing. The meeting would have run five minutes longer. The architect wasn't in the room. Whoever was would have suggested I take it offline, which is what I'm doing anyway."

She was right. Bringing the question up in the meeting wouldn't have produced an answer. It would have produced a procedural note that the question existed and would be addressed somewhere else. The

meeting wasn't a place where her question could be resolved. It was a place where she'd be told to resolve it elsewhere.

So she did. The question got answered properly two days later, in a corridor conversation with the architect. The team's velocity that sprint was unaffected. The standup stayed punctual and pleasant.

What had happened in that meeting, and others like it, was a small silent decision that nothing important was going to be said in it. No one had decided that explicitly. The meeting had simply taught the team, over months, that important things did not get resolved there. The board got updated. Procedural blockers got noted. The actual coordination work of the team had moved into the corridor, into Slack threads, into the quiet half-hour at the end of the day when the architect answered direct messages.

The standup had succeeded at what it had become, which was producing the appearance of coordination without the cost of it. Everybody in the meeting knew the meeting wasn't carrying the work anymore. Nobody said so, because saying so would have created a problem, and the problem of an empty standup is harder than the problem of any individual issue it could have surfaced.

By now I recognise the texture of a standup that has gone hollow. The updates are smooth. The Scrum Master's prompts are gentle and procedural. The board, if there is one, gets updated as people speak. There are no surprises and there is no friction. People look at the screen instead of at each other.

Compare that with a standup that's still doing its job. There is always at least one piece of friction. Somebody mentions a thing the rest of the team didn't know about, and the rhythm of the meeting breaks for

a moment while two people work out what it means. Someone asks a question that produces a real answer instead of a procedural one. Somebody admits they thought a piece of work was further along than it apparently is. The meeting runs four minutes over, and the Scrum Master isn't unhappy about that, because the four minutes were the part of the meeting that did the work.

The difference between the two isn't skill, but rather whether the meeting has accumulated enough cost on the people who say something difficult that they've learned, individually and quietly, to stop saying it.

In the team I was working with at the time, I tried something I have done several times since. At their next sprint review, instead of asking the usual question about whether they had hit their goals, I asked a different one.

What is not done that we are pretending is done?

The first time, silence. Not awkward silence. The honest silence of a question the team had not been asked before in that forum. The question was outside the repertoire. People didn't know how to answer it without feeling like they were doing something wrong.

The second time, two weeks later, a tentative answer. The tests for the story we closed on Thursday weren't complete; they were passing on a subset. The third time, a third answer, and a small conversation about what "Done" was supposed to mean and what it had quietly come to mean over the past six sprints.

Nothing dramatic happened. The team didn't transform. What changed was that the question existed in the room. After three sprints of asking it, the team started to answer it before I asked. Some of the

answers led to small process changes, some led to nothing. The point of the question wasn't to extract better information from sprint reviews. The point was to make truth slightly less expensive in the room.

> **What a working standup sounds like**
>
> **A standup that's still doing its job has a particular sound. Someone says something they hadn't planned to say. Someone asks a follow-up that isn't a clarification. Someone admits, briefly, to not knowing. Plans change in the room, even slightly. If your standups haven't sounded like that in a while, the meeting has stopped carrying its load, and adding structure to it usually makes the silence quieter, not louder. The thing worth knowing is what the silence is protecting people from. Pick one team member who has been quiet recently, and ask them, somewhere that isn't the standup, what they would say if saying it cost nothing. The answer is the diagnostic.**

TRY NEXT

Pick one of the four signals. Find one specific person who would benefit from being heard on it. Have a five-minute corridor conversation with them before the next standup. Ask: is there a thing you would say in standup if it wouldn't cost you anything? You will get an answer. The answer will tell you what the meeting is no longer carrying. You don't need to fix it in the next meeting. You need to know what it is.

A few weeks after that morning conversation with Mei, I sat in another team's standup, half-listening, watching the same texture I had begun to recognise. A junior developer was speaking. He mentioned, in passing, that he was waiting on a decision from one of the architects.

The Scrum Master nodded. The team moved on. The board did not change. The standup ended on time.

Walking out, I asked the junior developer what decision he was waiting on.

He told me. It was a decision that should have been made two weeks earlier and that was now blocking three other people, although nobody in the meeting had known it.

"Why didn't you say so in standup?" I asked.

He looked at me with the particular polite confusion of a junior person being asked a question to which the answer is obvious. "I did say so," he said. "I said I was waiting on a decision."

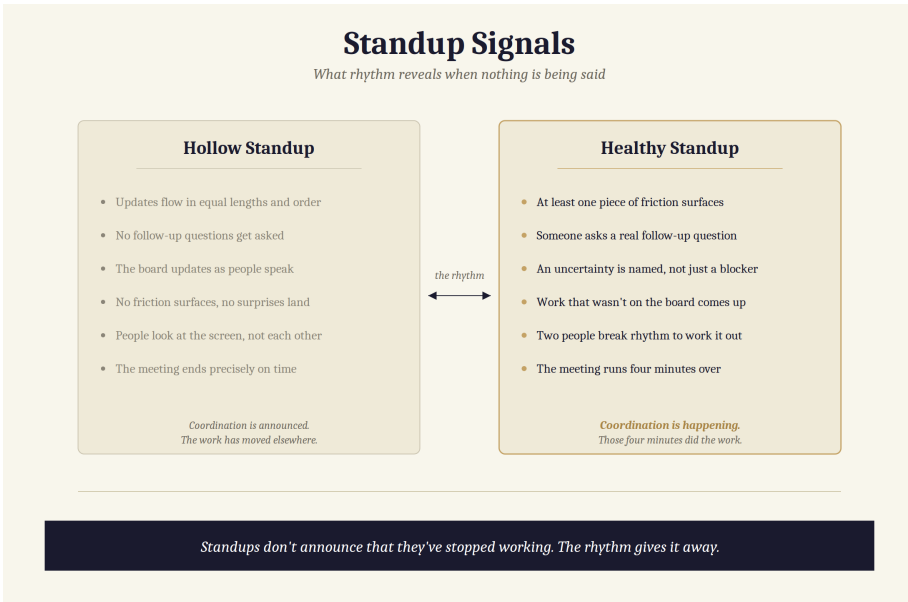


Figure 1.1: Standup Signals