Joe Hochrein

The FLOW Guide

The Definitive Guide to FLOW:
A Continuous Delivery Operating Model Built on Agile Principles

May 2026

Table of Contents

What is Flow?

FLOW is a continuous delivery operating model. It provides a lightweight framework that helps software teams deliver value continuously through disciplined execution, shared accountability, and constrained work in progress.

This guide is the authoritative definition of FLOW. Stages, accountabilities, and agnostic tooling adapt to fit any context. What cannot be adapted are the principles: work is pulled, work in process (WIP) is constrained, quality is a gate, and flow is visible. Abandon those and the result is the same as before: too much started, too little finished, and bottlenecks invisible until they become crises.

This guide is purposefully lean. It describes what the model is and how to run it. Specific tooling configurations, team-level tactics, and org-specific adaptations are context-dependent and outside the scope of this guide.

A Note on Scalability

FLOW scales without breaking. A single engineer can practice FLOW today using a personal board and a prioritized list. A full team with dedicated roles practices the same model with the same principles. The accountabilities, practices, and metrics do not change as the team grows. Only who holds each accountability changes. The same guide, the same board structure, the same practices apply from day one to full maturity.

A Note on Overhead

FLOW removes overhead by design. There are no required ceremonies beyond the refinement meeting. There are no mandatory roles beyond the four accountabilities. There are no sprint commitments, no velocity calculations, no capacity planning sessions, and no story point debates. What remains is the work itself, moving continuously from idea to delivered value.

Removing overhead does not mean removing discipline. It means replacing the wrong kind of discipline. Sprint commitments are replaced by pull-based execution and WIP limits. Velocity is replaced by cycle time and throughput. Capacity planning is replaced by aging indicators and real-time board visibility. The accountability is still there and it is more visible than ever.

A Note on Culture

FLOW is not just a delivery model. It is a cultural shift. Removing sprint commitments eliminates the shame of carryover. Pull-based execution gives engineers agency over when they start work. System signals replace personal blame when things stall. Swarming builds a culture of collective ownership rather than individual competition.

These are not soft benefits. Research consistently links psychological safety to higher team performance, lower attrition, and better quality. FLOW is designed to create that environment structurally, not aspirationally. The practices are the culture.

A Note on Scope

FLOW is designed for software delivery. Its stages, practices, and accountability model are built around the software development lifecycle: from requirement to build to release. Organizations may find FLOW principles applicable elsewhere but this guide addresses software engineering teams specifically.

FLOW Definition

FLOW is a continuous delivery operating model built on three ideas:

1. Work is **pulled** by those who have capacity from a prioritized backlog.
2. Work in progress is **limited** so that finishing is always prioritized over starting.
3. Delivery is **continuous**, decoupled from any calendar, ceremony, or time-box.

In a nutshell, FLOW requires a context where:

1. A prioritized backlog is maintained and ordered by value at all times.
2. Work is pulled from the backlog in highest priority item order.
3. Work flows through stages with clear accountability at each stage.
4. Flow health is continuously inspected and constraints are removed as they surface.
5. Software is released when it is ready.

FLOW Theory

FLOW is founded on lean thinking and systems theory.

Lean thinking reduces waste and focuses on the essentials. FLOW applies lean thinking through two core behaviors: **pulling** work only when there is capacity, and **constraining** how much work can be in progress at any time. Nothing sits idle consuming attention without delivering value. FLOW prioritizes **finishing** work over starting new work.

Systems theory recognizes that delivery speed is determined by the slowest constraint in the system, not by how hard individuals work. FLOW makes constraints visible through aging indicators and flow metrics so the system can be addressed as needed.

FLOW Principles

FLOW is built on eight principles. The first three describe how the system works. The last five describe how FLOW adopters behave. Together they give direction to every decision, action, and behavior in FLOW.

Visibility. WIP, stage accountability, and aging are visible at all times. Decisions are made based on what is actually happening.

Flow. Work moves continuously from intake to delivery. Nothing sits idle. Nothing is hidden. Bottlenecks surface immediately and trigger a corrective response.

Accountability. Each stage has a clear accountability. When work stalls, the system tells you where it is and what needs to happen next.

Finishing. Completing WIP is always prioritized over starting new work. A partially finished feature delivers no value. Done delivers value.

Transparency. The state of all work is visible at all times. Problems are surfaced as they happen, not discovered after the fact.

Ownership. FLOW adopters take ownership of their stage and do not pass problems silently downstream. When something is unclear, they surface it. When something stalls, they name it.

Quality. Nothing moves forward without meeting its exit criteria. Quality is not a phase. It is a gate at every stage.

Collaboration. When work stalls, those with capacity move to unblock it. Individual throughput is never more important than collective delivery. Finishing shared work is always more valuable than starting individual work.

FLOW creates a culture of continuous delivery built on collaboration, not ceremony. FLOW adopters finish together, ship with confidence, and stand behind every release.

FLOW Accountabilites

FLOW defines four accountabilities, not four required job titles. On a large team accountabilities map to dedicated roles. On a small team they compress onto fewer people. FLOW works at every scale.

Prioritization. Someone who maintains the backlog and decides what gets worked on next.

Requirements. Someone who defines what done looks like and validates that the work meets that definition.

Building. Someone who writes the code, tests it, and meets the exit criteria for Build.

Business Acceptance. Someone who confirms the work delivers the intended value.

FLOW Stages

Work moves through FLOW in a defined sequence. Every stage has a clear accountability, entry criteria, and exit criteria. Nothing moves forward without meeting exit criteria.

FLOW requires three stages at minimum:

Backlog A prioritized, ordered queue of work waiting to be pulled. Nothing leaves the backlog until it meets the Ready to Pull criteria.

Build Active WIP. Someone is building, testing, and meeting exit criteria. WIP limits apply here.

Released Done. Value delivered. Code merged, work closed.

What happens between Backlog and Build, and between Build and Released, is defined by the organization. Teams add stages based on their context, quality requirements, and team size.

Every stage an organization adds must answer three questions:

1. Who holds accountability for this stage?
2. What must be true for work to enter?
3. What must be true for work to leave?

If a stage cannot answer all three, it should not exist.

FLOW Practices

Pull-Based Execution

Work is never pushed onto engineers. Engineers pull the next highest priority item from the Ready stage when they have capacity.

Before pulling new work, every engineer asks in order:

1. Is there anything to **swarm** in Build aging or stuck that I can help move?
2. Is anything in Acceptance Testing that needs my support to unblock?
3. Only if both answers are no: pull the next backlog item.

This sequence is non-negotiable.

Pulling from the top, not by preference

The backlog is ordered by business value. Engineers pull from the top, not from wherever looks interesting or easiest. Cherry-picking tickets undermines the entire value of a prioritized backlog. The backlog order is the plan. Pulling from anywhere else is not FLOW.

When the Ready Queue Runs Low

When the Ready queue runs low, a refinement meeting is pulled together. A depleted Ready queue is a system signal, not a team failure.

Urgent and Unplanned Work

Emergencies, production incidents, and leadership-driven priorities do not bypass FLOW. They enter the backlog at the top. The next engineer with capacity pulls from the top as always, and urgent work is simply what is there.

When something cannot wait for natural capacity, whoever holds the Prioritization accountability directs an engineer to pull it immediately. That engineer's current ticket begins to age. The aging indicator fires and the team responds with a swarm or takeover. The disruption is visible on the board and the team absorbs it using the same mechanisms they use for any other stall.

As AI-assisted development accelerates the rate of code production, security vulnerabilities and unexpected discoveries will surface more frequently and with greater urgency. FLOW's urgent work mechanism handles these without special rules. They enter the backlog at the top and are pulled immediately.

FLOW does not pretend emergencies do not happen. It makes their impact visible rather than hiding it inside a sprint.

WIP (Work in Progress)

WIP limits are soft targets, not hard enforcement. The underlying principle is constant: **finish before you start.**

The recommended soft ceiling is 2-3 active tickets per engineer. Complex or high-risk work warrants fewer. The behavior matters more than the number. Most boards will not yet allow you to limit WIP per engineer, so this is a discipline that must be learned.

Aging indicators surface bottlenecks without restricting flow. Items exceeding the threshold are flagged on the board. The thresholds below are a starting point. Organizations should adjust them based on their own cycle times and add additional stages with thresholds that fit their context. The following table contains a starting point only, fine tune for your team.

Stage	Aging Threshold	Signal
Refinement	> 30 days	Requirements may be stuck or unclear
Backlog	> 60 days	Over 30 days may be a backlog that is too large
Build	> 3 days	Work may be stuck or blocked or ticket scope is too large
Acceptance Testing	> 3 days	Business acceptance testing is a bottleneck

Collaborative Work

FLOW recognizes three modes of collaborative work:

- **Swarming** is the default response when work is stuck or aging. A lightweight assist. The original engineer retains the ticket. Signal a swarm by adding a comment to the ticket noting what you are blocked on or ticket ages beyond agreed threshold in build stage. Swarming is not optional when the board shows aging work and you have capacity.
- **Co-Development** is intentional collaboration on something complex, risky, or high-value. Two developers work together with shared context, whether through modern IDE collaboration tools, screen sharing, or side by side. Use it when the work carries enough risk or complexity that a second set of eyes materially changes the outcome. It is not a rescue and not a default. It carries a real cost in throughput.
- **Takeover** is the transfer of full ticket responsibility from one engineer to another. Used sparingly. Always requires a full brief. No cold handoffs. Reassigning within the same stage does not reset the aging clock. The incoming engineer inherits the urgency. When a manager initiates a takeover it should be driven by delivery need, not performance management.

Situation	Mode
Ticket is aging or engineer is stuck	Swarm
Ticket is complex or high-risk from the start	Co-Develop
Engineer is blocked and swarming has not resolved it	Escalate to Co-Develop
Engineer is unavailable or cannot complete the work	Takeover
Junior engineer needs active guidance	Co-Develop

The goal is never individual output. The goal is delivered value.

Quality

Quality in FLOW is not a phase. It is a gate at every stage. Nothing moves forward without meeting exit criteria. Gaps found late in the process indicate a quality problem upstream, not a failure at the current stage.

Ticket Sizing

FLOW does not use story points or time estimates. FLOW uses abstract sizing. The only question that matters is whether a ticket is small enough to flow through Build without stalling.

Size	Meaning	Action
Small	Completable in a day or two	Ready to enter the workflow
Medium	Takes a few days, fits within normal flow	Ready to enter the workflow
Large	Too big to flow without risk of stalling, exceeds aging limits.	Break into smaller tickets under a feature flag or epic

Sizing is a flow decision, not a performance benchmark. A single ticket taking longer than expected is a system signal, not a performance signal. Performance is evaluated through patterns over time: throughput trends, cycle time consistency, swarming behavior, and quality signals. FLOW makes these patterns visible in ways that velocity never could.

FLOW Artifacts

The Backlog

A prioritized, ordered list of all work to be delivered. The single source of truth for what gets pulled next. Maintained by whoever holds the Prioritization accountability. Backlog order is driven by business value, not by engineer preference.

Tickets are preferably assigned before they reach Ready. Assignment happens during refinement, where engineers naturally volunteer for upcoming work or are matched to tickets based on expertise. Once assigned, the assignment is the plan. Engineers pull their assigned ticket when it reaches the top. Pulling someone else's ticket or skipping to easier work is not FLOW.

The Board

The real-time visual representation of all work in progress. The board columns map directly to FLOW stages. The tool-agnostic board IS the workflow.

The Release Queue

Done work accumulates in a Release Queue as tickets reach done status. The queue builds continuously until there is enough value to warrant a release. Value is determined by whoever holds the Prioritization accountability, based on business impact, customer need, or strategic timing. The decision to release is made in consultation with whoever holds the Business Acceptance accountability and is always based on readiness, not a calendar date.

Flow Metrics

FLOW replaces velocity with flow-based metrics.

Metric	What It Tells You
Cycle time (Build)	How long active work takes
Cycle time (Acceptance testing)	Where business bottlenecks live
Throughput	Items Done per week
Aging WIP	Where work is stalling
WIP Counts	Trending WIP over time tells you whether discipline is holding

Acceptance testing rejection rate	Quality of upstream requirements
Lead time	Total time from when a ticket enters the backlog to when it is released
Release Frequency	FLOW is built around continuous delivery so this is a natural health signal.

What Happens to Scrum Ceremonies in FLOW

In FLOW, meetings are pulled, not planned. The board is the signal. A stalled ticket, a depleted Ready queue, or a pattern worth examining pulls a conversation. When the board is flowing, there is nothing to meet about.

Meetings	**In FLOW**
Sprint Planning	Eliminated. The backlog is continuously prioritized. Engineers pull when ready.
Capacity Planning	Eliminated. Capacity is expressed through WIP limits.
Estimation / Story Points	Eliminated. FLOW uses abstract sizing.
Daily Standup	Replaced by board visibility, ticket comments, and aging indicators. The board tells the story the standup used to tell.
Sprint Review	Replaced by continuous release. Demos happen on demand and transparency happens through reporting.
Sprint Retrospective	Replaced by **SPAR** (Signal, Problem, Action, Result). A pulled meeting triggered by a system signal, not a calendar date.
Backlog Refinement	Retained. Pulled when the Backlog needs restocking. No time-box or schedule.

The Refinement Meeting

The one meeting FLOW explicitly retains. Its only job is to keep Ready work available to pull.

Who triggers it: Whoever holds the Prioritization accountability monitors the Backlog and calls a refinement session when the Backlog runs low. A healthy Backlog holds approximately 6 to 10 tickets per active developer. When it drops to 3 or fewer per developer, a refinement session should be pulled.

Who attends: Whoever holds the requirements and prioritization accountabilities. Engineers are welcome and often add value, particularly when volunteering for upcoming work or surfacing technical constraints. Discovery and ideation should happen upstream of refinement.

Cadence: Pulled, not planned. Triggered by the state of the Backlog, not the calendar.

What happens: Review upcoming backlog items, validate acceptance criteria, resolve open questions, size work, confirm Ready to Pull criteria are met.

What does not happen: Refinement is not sprint planning. Delivery commitments, scope negotiation, and status updates on work already in progress do not belong here.

SPAR: Signal, Problem, Action, Result

FLOW's replacement for the sprint retrospective. A pulled, blameless meeting triggered by a system signal, not a calendar date.

Signal. What triggered this SPAR? Always a system observation, never a person.

Problem. What actually happened? Walk through events without assigning blame.

Action. What will we change? Concrete, implementable actions with a clear owner.

Result. How will we know it worked? Define success before the SPAR ends.

Who calls it: Whoever holds the Prioritization accountability or the tech lead. If the process failure directly involves whoever would normally call it, the other steps in. The facilitator should have enough distance from the signal to keep the conversation focused on the system.

When to call a SPAR: When a pattern surfaces or when a single incident warrants a formal conversation. Not every bug or aging ticket requires a SPAR. Reserve it for signals worth a structured response.

Output: A written summary covering all four parts is added directly to the ticket that triggered the SPAR. Over time the ticket history becomes a living record of how the team responded to and improved its system.

FLOW Adoption

You do not sell FLOW. You demonstrate it.

FLOW does not require a big rollout or a team-wide process change. Adoption starts with a single engineer today and spreads naturally because the behavior is visible and the results quickly follow.

Starting with a Single Engineer

A standard Kanban board is all you need to start. Three columns: To Do, In Progress, Done. One engineer, one prioritized list, three behaviors:

1. Pull one ticket at a time. Do not start a new ticket until the current one is done or genuinely blocked.
2. Before pulling new work, check whether anything is aging or stuck and help move it first.
3. Add a comment to a ticket when blocked rather than sitting on it silently, ask for Swarm.

No process change is required from anyone else. These three behaviors practiced consistently will become visible to teammates and managers within weeks. That visibility is what opens the door to team-level adoption.

Expanding to a Team is straightforward:

- Agreement on the board stage structure
- Soft WIP targets visible on the board
- Aging indicators configured in the tool

Setup in as little as a half-day, not a month-long transformation.

Expect Resistance

Resistance is normal and should be expected. Engineers comfortable with Scrum may miss the structure of sprint commitments. Managers attached to velocity reporting may feel they are losing visibility. Neither concern means FLOW is wrong. It means the transition needs patience and evidence.

The answer to resistance is never argument. It is continued demonstration. Cycle time improves. Tickets finish. Quality signals get cleaner. The board tells the truth. Let the data respond.

What FLOW Is Not

- **Flow is not a rejection of Agile. It is a maturation of it**
- **Flow does not lack structure.** The prioritized backlog, WIP limits, aging indicators, stage accountability, quality gates, and SPAR practice provide more visible structure than most frameworks deliver.
- Flow is not a framework that requires perfection to start. One engineer, one board, one prioritized list is enough.
- Flow is not for every context. FLOW requires a backlog that is genuinely maintained and prioritized, validation that the business will participate in, and leadership willing to measure flow health rather than sprint commitments. Without those three conditions the model will not hold.

Definitions

Abstract Sizing Not estimation. It is a triage decision. The only question is whether a ticket is small enough to flow through Build without stalling. Small, Medium, and Large are not effort forecasts. They are flow signals. Large means break it down before it enters the queue. Abstract sizing is a flow decision, not a performance benchmark.

Acceptance Testing The business-owned confirmation stage. A first-class constraint in FLOW, not an afterthought. Acceptance Testing is validation that the work delivers the intended value, not discovery of new requirements.

Aging Indicator A visual signal on the board flagging work that has exceeded its stage threshold. Surfaces bottlenecks without restricting flow.

Backlog A prioritized, ordered list of all work to be delivered. Tickets are preferably assigned before they reach Ready. The single source of truth for what gets pulled next. Backlog order is driven by business value, not by engineer preference or assignment.

Build The active development stage. Held by whoever is doing the technical work from pull to exit criteria met.

Co-Development Intentional collaboration between two engineers on complex, risky, or high-value work. Not a rescue. A strategy.

Cycle Time The elapsed time a ticket spends in a given stage. The primary indicator of system health.

Done A ticket is done when it has met all quality gates required to exit its current stage. In FLOW, done has two levels. Build Done means code is complete, tested, peer reviewed, and meets all acceptance criteria. Released means the work has been accepted in Acceptance

Testing, merged, and delivered. A ticket is not done because an engineer says it is done. It is done because the exit criteria say it is done.

Exit Criteria The criteria a ticket must meet before moving forward. Two levels: Build Done and Released.

FLOW A continuous delivery operating model and lightweight framework built on pull-based execution, constrained WIP, and shared accountability. Work moves continuously, without ceremony, without shame, without artificial stops.

Lead Time The total elapsed time from when a ticket enters the backlog to when it is released. Where cycle time measures a single stage, lead time measures the full delivery journey.

Pull The mechanism by which work and meetings happen in FLOW. Both are triggered by need and capacity, not by a calendar or a plan. If there is no signal and no need, there is no meeting and no new ticket.

Ready to Pull Criteria The criteria a ticket must meet before it can be pulled into Build. Includes clear acceptance criteria, appropriate sizing, and no open questions that would block an engineer from starting.

Release Queue The accumulation of completed work that builds as tickets reach Released. Releases are drawn from the queue when there is sufficient value, not on a fixed schedule.

SPAR (Signal, Problem, Action, Result) FLOW's replacement for the sprint retrospective. A pulled, blameless meeting triggered by a system signal. Not a calendar event.

Swarming The behavior of converging on stuck or aging work rather than starting new work. Both a workflow practice and a cultural expectation.

Takeover The transfer of full ticket responsibility from one engineer to another. Used sparingly, especially as a management action.

Throughput Items completed per unit of time. Combined with cycle time and lead time it gives a complete picture of flow health.

WIP (Work in Progress) Work that has been started but not finished. High WIP is the enemy of flow.

WIP Limit A soft constraint on active work per engineer or per stage. Enforced through behavior, not system lockdown.

