PLAYBOOK · MAINTAINED REFERENCE

The Planning Playbook

Ed Schaefer · Version 1.0 · August 2026 · edward-schaefer.com/playbooks/planning · Free to share

Most teams go through the motions in sprint planning. They pull stories into the sprint with little discussion, or worse, get handed a list: "here's what's in the sprint, we're gonna start."

Planning is the last event in the cycle's sequence: the review closes out the last cycle, the retro follows it, and planning comes next. By the time planning starts, the team already knows what finished, what didn't, and why.

This document works two ways: read it start to finish and run a real planning session this cycle, or keep it open as a live reference once the shape is familiar. Point an agent at it and it can do either job.

1. The 60-second version

- **Scales up and down.** The same shape works for a two-week sprint and a quarterly plan. Only the event's size changes.
- **Starts with review already closed.** Spillover from the last cycle comes in as input, ready to use. Planning doesn't start from scratch.
- **Default play:** pull the sprint's candidates, revalidate each one's value, break work into tasks where it's genuinely useful, then the product role frames the vision and direction, plus any feedback that's come in.
- **Points, if used at all, are a too-big check,** nothing more. The team picks its own sizing approach.
- **Definition of Ready is a filter that helps confirm work can actually start.** It doesn't block urgent work from starting anyway.
- **Duration:** ninety minutes is the sweet spot for remote teams, an hour is doable but tight, two hours is the real ceiling, thirty rarely covers it.
- **With agents:** plan more often, in smaller pieces, and don't story-point the work an agent is doing.

2. The default play

The ladder starts with review already done. If it isn't, the team closes out the last cycle first, and it eats into planning's own time.

Spillover comes in as input. Whatever didn't close last cycle carries forward as the starting list.

Pull and revalidate. As each candidate item gets pulled toward the sprint, the team checks it's still the right next thing: still valuable, still correctly understood. The backlog's own ordering doesn't answer that question by itself.

Task-level breakdown, where it earns its keep. The story stays the user-facing outcome; the tasks underneath it are the technical steps it actually takes to deliver, so people can chip away at something concrete instead of one large story. This stays at team and engineer level. Formal splitting technique is a separate topic entirely, deliberately out of scope here (the table in Section 5 draws that line). It helps most with mixed seniority in the room, when a junior engineer or an intern benefits from talking it through more than a room of only senior engineers would need.

The product role's beat. Vision and direction, any updates to either, business needs, and feedback that's come in: the overall picture, delivered by whoever holds the product role. I've been in rooms where this ladder runs the way it's supposed to, and everyone leaves with the same understanding of what's next and why.

Who runs it. A facilitator, making sure the team goes through the steps and stays aligned. Same default as the review and the retro: the scrum master or team coach, the product owner, or some combination of the two — any of them can hold it here too.

One way this ladder breaks is overcommitment, and it's a big one. Section 3 exists to fix it.

3. The pull-then-trim mechanic

This mechanic catches overcommitment before the sprint starts instead of mid-sprint.

Before the session. Pencil a future sprint ahead of time, without knowing yet whether it can actually be done. It's a rough draft, nothing more, usually overloaded on purpose. At review, once current-cycle items are closed out (backlog, next-sprint, or done), and before spillover rolls in, walk the penciled draft and prune or add to it. Spillover rolls in after that. Refinement happens separately, so the backlog itself doesn't crowd into planning.

Then the team trims. Usually I've overloaded the sprint on purpose, and I ask the team to pull things out and tell me what they can't do, and we keep trimming until they feel good about what's left.

The rule underneath it: if it's ever a real question whether something fits, pull it out. It can always come back into the sprint, or start early if capacity opens up, and that's better than overcommitting.

One caution: watch for sandbagging, padding the trim so nothing ever feels tight. The goal is a plan the team is genuinely confident in. Go with what you're confident about, and then you can always pull more in.

Where the sources disagree

Whether planning is a negotiation at all is a live, unresolved split in the practitioner literature. One camp frames it as a commitment negotiation among the team, the product owner, and the organization. A more recent camp rejects that frame for the capacity question, arguing the team should simply assess what's achievable rather than negotiate it, while still leaving room to negotiate priority.

Both positions are argued seriously. This playbook doesn't take a side on that question; the mechanic above works under either framing.

4. Script: running planning

Spillover.

> "Here's what's carrying over from last sprint. Let's confirm each of these before we add anything new."

Pull and revalidate.

> "Is this still the right next thing? Has anything changed about its value, or how we understood it?"

Task breakdown.

> "Let's talk through how we'd actually deliver this as a team. What are the pieces?"

The product role's beat.

> "Here's where things stand: the vision and direction, plus any feedback since the last sprint."

The trim.

> "I've probably overloaded this sprint on purpose. Pull out what you don't feel good about, and tell me what you can't do. We'll keep trimming until it feels right."

Close on confidence.

> "We don't need everything in. Go with what you're confident about. We can always pull more in later."

If you're not the one running it. Planning that's gone through the motions doesn't fix itself; asking for better is a script too, built from the failure modes in Section 8 and the trim discipline in Section 3:

> "I think our planning could use a couple changes. Right now it feels like we mostly get handed the sprint instead of talking it through. Could we make it more of a conversation? When we pull something in, could we check out loud whether it's still the right next thing, instead of just taking whatever's next on the backlog? For anything that isn't obviously small, a few minutes talking through the tasks would help too. And if something's a stretch, I'd rather say so in the room than find out mid-sprint."

5. What belongs, what doesn't

Segment	Belongs	Doesn't belong
Spillover	Confirming carried-over items are still right before adding anything new	Treating spillover as a surprise, or skipping the confirmation
Pull and revalidate	Checking value and understanding on every item pulled in	Auto-pulling whatever's next in backlog order without discussion
Task breakdown	Team-level breakdown for work that benefits from it, especially with mixed seniority in the room	Formal splitting technique: vertical or horizontal slicing, spike-versus-build calls. A different topic entirely
Product role's beat	Vision, direction changes, business needs, feedback received	A plan handed down with no discussion or evaluation
The trim	Pulling items out until the team feels confident, guided by the uncertainty rule	Sandbagging the trim, or overcommitting because nobody speaks up

6. Estimation

Points, if a team uses them at all, are a too-big check and nothing more. In my ideal world, that's the only job they do: catching a story that's actually multiple stories before it gets pulled in. Beyond that, my own lean is toward throughput: counting finished work and turning it into probabilistic forecasts instead of a per-story number. Don Reinertsen, Troy Magennis, and Dan Vacanti have built a mature body of practice here, influential for a long time, if still not widely adopted outside teams that go looking for it.

None of that makes points wrong for every team. This playbook works for whatever approach an organization actually runs, and that's the team's own call. Say-do ratios are the same story: useful if the organization genuinely needs that signal, skippable if it doesn't.

Picking a sizing approach in depth is its own topic. What belongs here is smaller: don't let points become more than the too-big check, and pick a sizing approach deliberately.

Where the sources disagree

Story points versus "#NoEstimates" is a real debate, more than a decade old, and no research in wide circulation settles points versus time versus no estimate. My own default stands: dropping story-pointing with agents (Section 10) is my own call inside that open debate. No research settles it either way.

7. Definition of Ready

A Definition of Ready is a short checklist a team agrees on for what has to be true before an item gets pulled into a sprint: approvals in hand, dependencies identified, enough shared understanding to actually start. Some teams swear by one; some teams have never used one.

My own approach: treat it as a filter, and only that. On complex work it earns its place, because it catches the item that can't actually start before anyone pulls it — nothing forces a mid-sprint "why did we even pull this in" conversation.

Weaponized, the same list turns into a rule that blocks urgent work from starting even when it can't wait, or when something's changed enough that starting unready is still the right call, with no exception for either.

8. Failure modes and fixes

The handed-down plan. "Here's the plan, and you're gonna deliver to it." Not much discussion, no real evaluation. Close enough to ProdPad's named anti-pattern, "Waterfall in Sprints," that it's clearly not just a personal impression. Fix: make planning an actual conversation. Nobody hands down the plan without discussion.

Skipping task breakdown. Partway through the sprint, the team realizes the work is bigger than they thought, something they probably could have caught in planning. Fix: light task-level breakdown for anything that isn't obviously small (Section 2).

Review content leaking into planning. When review doesn't actually happen, its content surfaces for the first time in planning instead. The Review Playbook names this same pattern as review's own clearest failure sign. Fix: close review before planning starts (Section 2); if it isn't closed, closing it is the first thing planning has to do.

Auto-pull from the backlog. Picking up whatever's next on the backlog without checking it's still the most valuable thing to do next. Fix: revalidate value on every pull (Section 2). Ask whether it's still valuable instead of assuming it.

Forcing the plan to hold. Treating the plan as fixed once it's set, as if it can't change. Fix: expect the plan to move, and revisit it as the sprint goes. The uncertainty rule in Section 3 assumes exactly that.

Overcommitting the plan. "Overloading the plan or overcommitting on the plan" is, in my experience, a big one. It's the textbook case of what Kahneman and Tversky named the planning fallacy, a documented tendency to underestimate how long work will take, and it's the specific anti-pattern Stefan Wolpers names Over-commitment in his own taxonomy of sprint-planning failure, independent confirmation this isn't just one team's problem. Fix: the pull-then-trim mechanic in Section 3 catches this before the sprint starts.

9. Variations

Duration. The Scrum Guide caps Sprint Planning at eight hours for a one-month sprint, and says shorter cadences run "usually shorter," without a formula; SAFe's own Iteration Planning timebox lands on a specific number instead, about ninety minutes for a two-week iteration. My own ladder: thirty minutes is rarely enough, maybe on a one-week sprint with a team that's really on top of things, nowhere else. An hour is doable, sometimes tight. Ninety minutes is my sweet spot, especially with remote and virtual teams. Two hours is breathing room, and I'd rarely go past it. That's my own calibration; the guide's own ladder feels about right to me, and I've found real value in letting a planning run long when the team uses the extra time for actual collaboration, even a pairing or mobbing plan made in the room.

Scales up and down. This applies even at the quarterly level: planning could scale up and down for sure. The ladder in Section 2 doesn't change shape; only the size of the horizon does. A quarterly plan pulls quarter-sized candidates and revalidates them the same way a sprint plan pulls stories.

Kanban and no-sprint teams. Some Kanban teams plan on a schedule; others plan ad hoc, as needed. The default holds regardless: pull and revalidate before committing capacity.

Frameworks without a discrete planning event. Not every framework has one; Extreme Programming folds planning into its continuous Weekly and Quarterly Cycles instead of a single scheduled meeting. What matters is whether the function happens somewhere. If a team already has a home for pull-and-revalidate and task breakdown, the job is covered without a meeting called "Planning."

10. AI-era

Plan more often, in smaller pieces. I'm planning a lot more frequently now, but the plans are also smaller, and I'm iterating on them a lot faster than before.

Stop story-pointing agent work. I basically don't do it at all with agents anymore; Section 6 stops applying once an agent is doing the work. What agents are actually good at is estimating in human-time instead: "this will be ready in two weeks" turns into something that takes twenty minutes. The unit of estimation has shifted.

Capacity is WIP-limiting me. What matters now is the chunk size: how much to hand off before I check in or steer it again. That chunk size is the real planning decision now, more than any point total ever was.

The spec-loop pattern. In practice: a hooks-driven loop that chunks its way down through a spec, making a fresh micro-plan at each step. The request format that drives it is outcomes and friction, not exact specs: what I want accomplished, and where I'm hitting friction trying to get there.

The caution that matters most. Repeatedly saying "what's next on the plan, keep working," without actually reviewing what came back, leads to a pile of work I don't really understand: so much has shifted that it's hard to tell if what I wanted is working well, or if it's very surface level on ten things instead of one thing working really well. Barely scratching the surface, across everything at once, is exactly what

more-frequent, smaller plans are supposed to prevent. Nobody Is WIP-Limiting the Human goes deeper on where that thread leads.

11. This is a fit question, not a recipe

The shape above is a strong default. Where a specific team actually sits against it, and what that's costing, is what the Engineering Ways of Working Diagnostic is built to find.