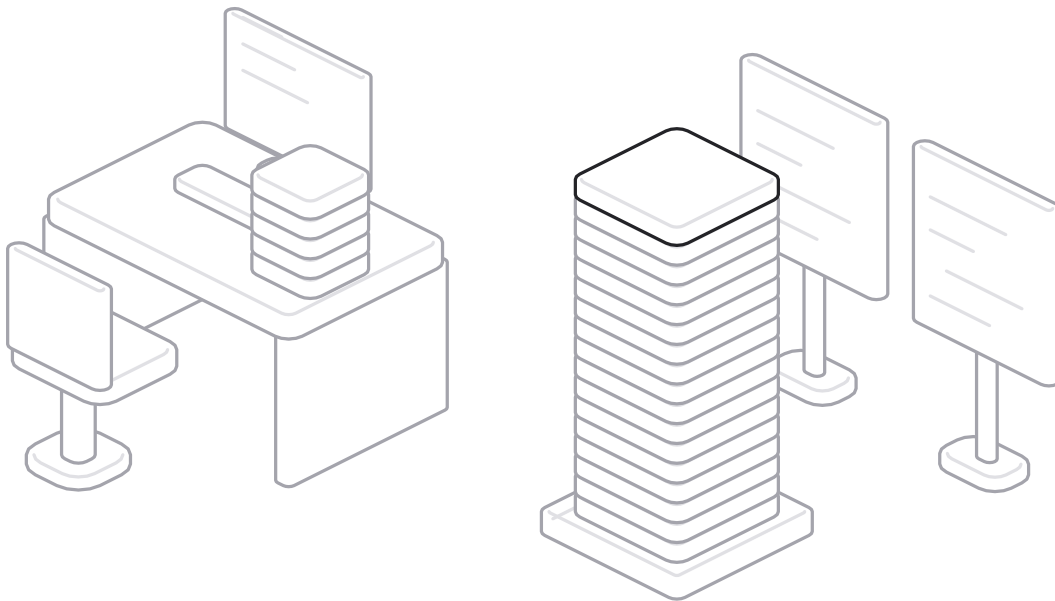


GUIDE

Raising team leverage

How engineering teams get more finished work from every hour of attention, with coding agents doing the building.

For engineering leaders whose teams already build with Claude Code, Cursor, or Codex.



Contents

	In short	3
01	What is engineering leverage?	4
02	Why hasn't AI raised our team's throughput?	5
03	Where does your team's attention go?	6
04	What are the five factors behind leverage?	7
05	How do you shorten the time from first draft to agreement?	8
06	How do you keep agent work from coming back?	8
07	How do you raise throughput without burying reviewers?	9
08	How do you stop agents waiting on people?	10
09	How does team knowledge raise leverage?	11
10	Do more agents running in parallel raise leverage?	11
11	How do you set a leverage goal?	12
12	Where Hamster fits.	13
	Common questions	13
	Sources	14

Web version, with the templates to copy: tryhamster.com/guides/team-leverage

In short

Engineering leverage is the finished work a team ships for each hour of attention its people spend, and the companion guide [How to show that AI is working](#) defines and measures it in full.

-
- Start by finding where the team's attention goes, split into briefing, steering, review, rework, and reviewing others, and take the largest part first.
-
- Coding agents often leave team throughput flat because the hours they save in writing code move to review, rework, and waiting on people.
-
- Five factors explain leverage. They are alignment velocity, decision quality, ship-ready rate, cycle compression, and context coverage.
-
- Raise the factor that drags most first, and change one factor at a time so the team can tell which change worked.
-
- Parallel agents raise leverage only when their work merges without pulling people back in.
-

Coding agents made code cheap to write, and many teams still ship at the same pace. The hours agents save in writing code go to briefing, steering, review, and rework instead, so the work moves to people rather than going away.

Leverage measures that shift. It is the finished work a team ships for each hour of attention its people spend, and coding agents should raise it. While attention piles up, leverage stays flat however busy the agents are. [How to show that AI is working](#) defines the measure in full and shows how to compute it and report it to a board.

Raising leverage works like any throughput problem. Find where the team's attention goes, find the factor that drags most, and change that one thing before touching the others. Five factors explain leverage, each with changes that move it, and a leverage goal lets the team read its progress every month. The method needs only records every team already has, a git host and its briefs or tickets. Raising leverage matters only when the customer or business outcomes the work targets move with it, so read leverage beside those results.

What is engineering leverage?

Engineering leverage is finished work per hour of attention. Finished work is work that merges without coming back, sized by the hours the team would have needed without agents. Attention is the time people spend getting it there.

Attention has five parts. Briefing is shaping briefs and plans before work starts. Steering is prompting agents, answering their questions, and correcting course while they build. Review is checking the work your own agents deliver, and rework is fixing work that came back after review. Reviewing others is time spent on teammates' pull requests, and it counts. Meetings and idle time do not count, although they cost attention too.

At 1.0x, an hour of attention shipped an hour of work, the same as building it yourself. At 3.4x, ten hours of attention shipped 34 hours of work.

Work is sized by what it would have cost to build, never by pull requests or lines, so a few pull requests that merge clean beat many that bounce back. Work that goes back for rework still counts once it merges clean. The hours spent sending it back and fixing it stay in the attention total, so bounced work costs twice. For 30 days after a merge, a revert cancels the work, and a follow-up fix charges its hours to the work it fixed.

TEMPLATE Leverage, defined

Leverage = finished work (team-hours) ÷ attention (hours), over a rolling window.

Finished work: work that merged without coming back, sized in the team-hours it would have taken without agents. Never pull requests or lines.

Attention: briefing, steering, review, rework, and reviewing others. Meetings and idle time are left out.

1.0x: an hour of attention shipped an hour of work.

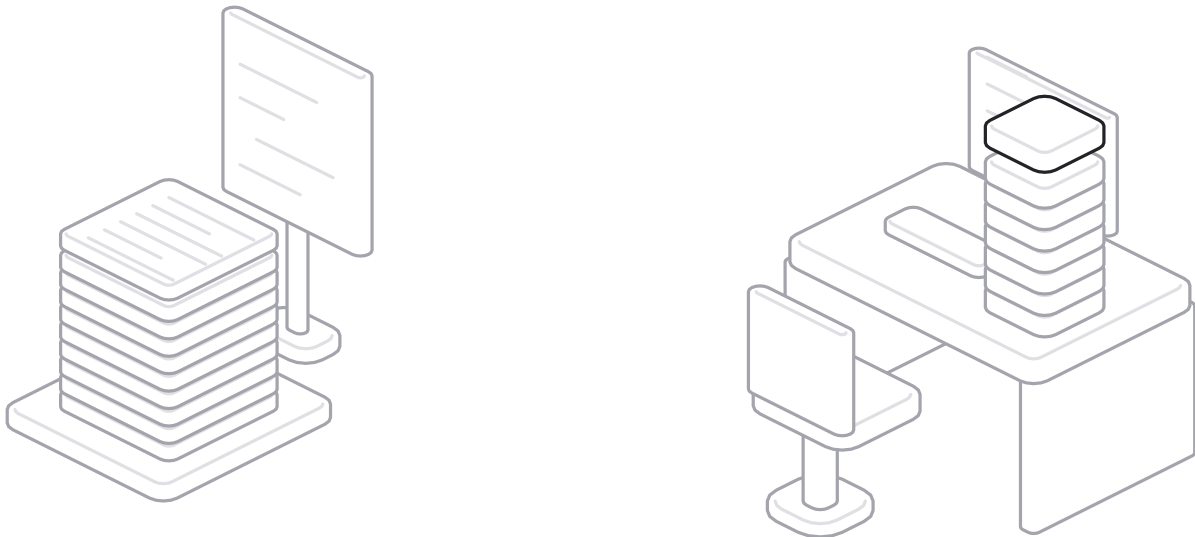
The 30-day tail: a revert within 30 days cancels the work and keeps its hours. A follow-up fix charges its hours to the work it fixed.

The team figure is all finished work divided by all attention. It is not the average of each person's figure.

Why hasn't AI raised our team's throughput?

Agents sped up a part of the work that was not the constraint. Code got cheaper to write, and the hours moved to review, rework, and waiting on people, so the team ships about as much finished work as before.

Faros looked at telemetry from more than 10,000 developers and found that on teams with high AI adoption, developers merged 98% more pull requests. Time spent in review rose 91%, and the average pull request grew 154%.¹ Pull requests and review time both nearly doubled. A leader who reads only the first number sees a team twice as fast. Leverage reads both, because review is attention.



Eliyahu Goldratt described the general rule in *The Goal*, his novel about a failing factory: "An hour lost at a bottleneck is an hour lost for the entire system. An hour saved at a non-bottleneck is a mirage."² With coding agents, the bottleneck is human attention. Agents have capacity to spare and people do not. An hour an agent saves in writing code counts only if it does not cost a person an hour somewhere else.

An hour saved at a non-bottleneck is a mirage.

Eliyahu M

A developer who finishes a feature in a day while the team takes three weeks to ship it is seeing the same effect. The day of building is fast. The weeks go to agreeing on what to build, rounds of review, and fixes after review. Each of those is attention, or a queue in front of a person.

03

Where does your team's attention go?

Split the last 30 days of attention into the five parts and look at the largest. The largest part points to the factors that would shrink it.

The [leverage worksheet](#) in the companion guide collects attention per person and per part from the timestamps your tools already keep. Run it for the team as a whole. Reviewing others lowers the reviewer's own figure on purpose, because it makes a lead's review load visible, and a ranking would punish the people who review.

Each part of attention has a cause you can work on. The map below names the factors that shrink each part, with the most direct one first.

TEMPLATE **Attention map**

If most attention goes to	Look first at	Why
Briefing	Alignment velocity, then context coverage	Briefs that align quickly need less back-and-forth to shape. Plans that draw on what the team already knows need less shaping.
Steering	Context coverage, then cycle compression	A plan grounded in what the team already knows answers the questions agents would otherwise ask. Agents that wait on people less finish before anyone has to step back in.
Review	Context coverage, then ship-ready rate	Work planned from what the team already knows arrives closer to done. Small pull requests that match the plan are faster to check.
Rework	Decision quality, then ship-ready rate	Settling open questions before delivery keeps work from coming back. Small pull requests that match the plan come back less often.
Reviewing others	Ship-ready rate	Teammates' small, plan-matched pull requests take less time to review.

Try it with your AI

Your coding agent can run this guide with you. Paste one line into Claude Code, Cursor, or Codex, in the repo your team works in.

Read <https://tryhamster.com/guides/team-leverage/skill.md> and run it with me.

1. It reads the last 30 days of merged pull requests and reviews from your repo.
2. It estimates attention by part and shows its working, with every assumption marked.
3. It names the factor that drags most and drafts one change to raise it.

04

What are the five factors behind leverage?

Five factors explain leverage: alignment velocity, decision quality, ship-ready rate, cycle compression, and context coverage. Each one comes straight from delivery records, and the one that drags most is usually the cheapest to move.

The factors follow the loop most teams already run. Discovery measures how fast a brief reaches agreement, and Direction measures whether the brief settled the questions that would otherwise come back as rework. Delivery covers how clean the pull requests are and how long agents wait on people. Knowledge measures how much of each plan the team's existing context already answered.

Read each factor against a fixed neutral point that you set once for your team, such as its own value in a baseline month. A factor above neutral is helping and a factor below it drags. The five factors explain leverage without computing it. If leverage moves and none of the factors moved with it, something you are not measuring changed, and that gap deserves its own line in the monthly read.

Change one factor at a time. A team that changes five things in a month cannot tell which change worked, and the factor that drags most is the bottleneck Goldratt warned about. Time spent on any other factor is a saving at a non-bottleneck.

TEMPLATE Five factors

Loop stage	Factor	Measure	Rises when
Discovery	Alignment velocity	Days from a brief's first draft to agreement	The right people weigh in early.

Loop stage	Factor	Measure	Rises when
Direction	Decision quality	Share of briefs whose work needed no rework	Briefs settle open questions before delivery.
Delivery	Ship-ready rate	Share of work merged without going back	Pull requests stay small and match the plan.
Delivery	Cycle compression	Hours from task created to merged pull request	Agents wait on people less.
Knowledge	Context coverage	Share of plan tasks grounded in existing context: a decision, a design document, a team skill, or a prior brief	The team's record already holds what a plan needs.

05

How do you shorten the time from first draft to agreement?

Name the people whose work a brief changes on the day it is drafted, and ask each of them to vote on the current draft. Alignment velocity rises when the right people weigh in early.

A slow agreement costs attention as well as days. The author answers the same questions in several threads and edits the scope again after each late reader. A short, explicit agreement step removes most of that work. The [alignment checklist](#) in Before the agents start lists what to settle before a brief goes to agents.

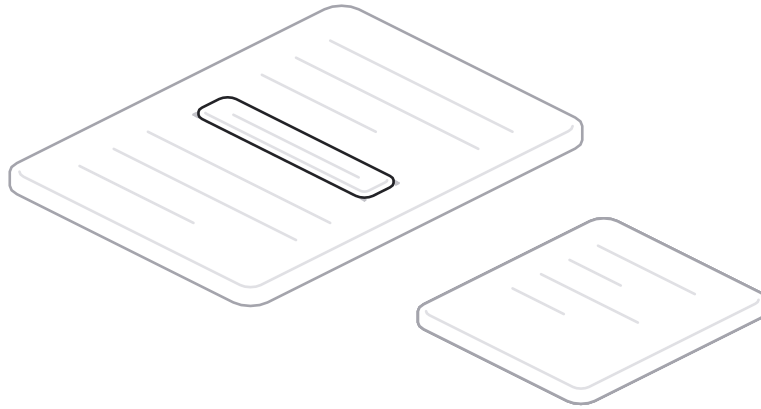
Measure the days from first draft to agreement for every brief in the month. If a few briefs account for most of the days, read them. They usually share a cause, such as an owner nobody named or a reviewer added late.

06

How do you keep agent work from coming back?

Settle the open questions in the brief before any agent starts building. Decision quality is the share of briefs whose work shipped without rework, and it rises when the brief left nothing important for the agent to guess.

An agent fills a gap in a brief with a guess. When the guess is wrong, the team pays for it after the pull request opens, in review comments, a second round, and a fix. Hamster's advice is to align on the brief first, because it beats arguing over a pull request.



Record why each piece of work came back, using three causes. The scope was wrong or missing, the work had a defect, or the work did not match the plan. If most rework traces to scope, the fix belongs in the brief. If most traces to defects, look at tests and at the size of each pull request.

07

How do you raise throughput without burying reviewers?

Keep each pull request small and tied to one task in the plan, and review it against the plan. Ship-ready rate, the share of work that merges without going back, rises when pull requests stay small and match what the team agreed.

Review gets worse as pull requests get bigger. In SmartBear's study of code review at Cisco Systems, reviewers were most effective on 200 to 400 lines at a time. Reviewing faster than about 500 lines an hour cut the number of defects they found.³ Agents push the other way, and on the high-adoption teams in the Faros data the average pull request grew by 154%.¹ A reviewer facing a large diff either slows down, which costs attention, or skims, which lets rework through to the 30-day tail.

Small batches pay off beyond review. DORA's research finds that working in small batches predicts both software delivery performance and organizational performance.⁴ For agent work, a small batch is one plan task per pull request, with the task named in the description. The reviewer then checks one intended change and does not have to reconstruct it from the diff.

The same rule answers a common question about running agents overnight. Starting the runs is cheap. Overnight work raises leverage only if the morning review is short and the work does not come back later. The 30-day tail shows whether it came back. If the morning goes to untangling large diffs, the overnight runs lowered leverage, however many pull requests they opened.

TEMPLATE Pull request rules for agent work

- One plan task per pull request.
- The description names the task and the brief it came from.
- Keep each pull request under 400 changed lines, excluding generated files.
- Review against the plan: did this pull request do its task, and nothing else?
- If review sends it back, record why: scope, defect, or plan mismatch.
- Charge any fix within 30 days of merge to the original work.

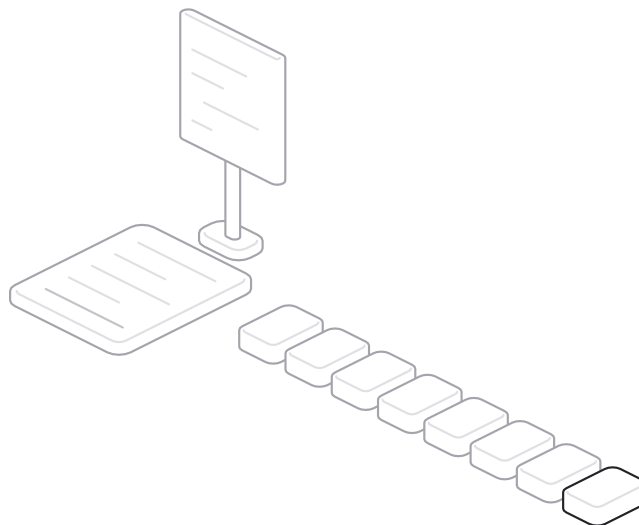
08

How do you stop agents waiting on people?

Answer agents' questions in the plan, before they ask, and keep review queues short. Cycle compression is the hours from a task's creation to its merged pull request, and it rises when agents wait on people less.

An agent that stops to ask a question waits until someone answers, and a finished pull request waits for its first review. Neither wait costs attention while it lasts, but both stretch the cycle, and a long cycle costs attention later. The person who answers after two hours has to reload the context. Work that sits for days drifts from the main branch and needs fixing before it merges.

Two habits shorten the waits. Put the decisions an agent would ask about into the plan, so the run never pauses. Then give reviews a fixed slot each day, so pull requests do not wait for someone to notice them.



09

How does team knowledge raise leverage?

Context coverage is the share of plan tasks that the team's existing context already answered, from a recorded decision, a design document, a team skill, or a prior brief. A grounded task needs less briefing, less steering, and less review. A task planned from nothing needs all three.

The record of what a plan could not find is as useful as the record of what it found. When planning a guest checkout turns up no decision about guest accounts and no design document for checkout, the gap is the next thing to write down. Each gap filled once saves the briefing, steering, and review it would have cost on every later piece of work that touches it.

Score the team's skills and shared instructions the same way. Compare the leverage of work that used a skill with the leverage of work that did not, over the last 90 days. A ratio above about 1.1 suggests the skill helps, and below about 0.9 suggests it holds work back. With fewer than three pieces of work on either side, there is too little to tell. The comparison shows what moved together and cannot show cause, so treat a low score as a reason to read the skill again.

In Hamster, the team's decisions and context live in the [Context Graph](#), and team skills live in [Skills and Methods](#). Hamster's planning searches the Context Graph before it writes each task.

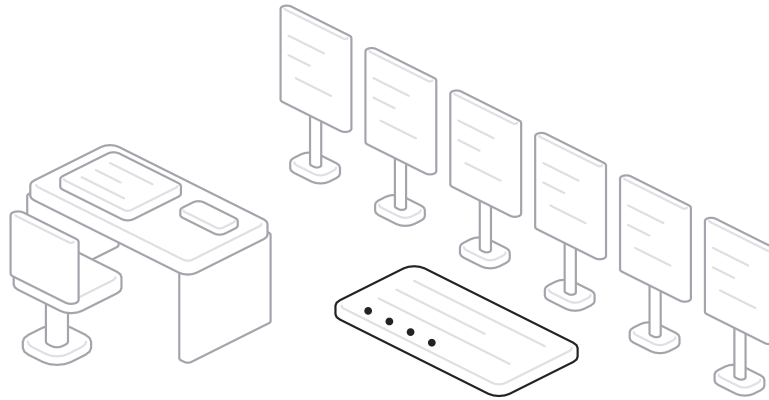
10

Do more agents running in parallel raise leverage?

Only when their work merges without pulling people back in. Parallel agents that each need steering and a long review keep the attention per hour of finished work high, so leverage stays flat however many sessions are open.

Leverage does not count sessions. Parallel work shows up as more finished work for the same hours of attention, or it does not show up at all. Take an engineer juggling six agent sessions, each of which asks questions and returns a large diff. Each finished hour costs as much attention as it would in one session, plus the cost of switching between them. The same six sessions, run from an agreed brief and a plan grounded in the team's context, can finish with little steering. Then the parallel work does raise the number.

Compare the two kinds of work directly. Measure leverage for work that went through your harness, the brief, plan, and team context your agents work from, and separately for work prompted directly. Your records will show less attention for the directly prompted work than it really took, because commits and reviews are visible and the prompting before them is not. Its leverage is therefore an upper bound. If the harness work still comes out ahead, the harness is paying for itself.



11

How do you set a leverage goal?

Record a baseline, set a target and a date, and tie the goal to the one factor you chose to raise. Read it monthly, as a team figure, and log every change to how work runs so you can see what moved with it.

A leverage goal is an ordinary goal with a baseline, a target in $\times$, a target date, and a note on whose work it covers. Track pace from the day tracking began to the target date, so a goal that is behind shows it early. Recent work keeps collecting follow-up fixes for 30 days, so the latest month settles over the next one.

Log every change to how the team works, such as a new skill, an edited design document, a new routine, or a changed delivery setting. Compare leverage for the two weeks before each change with the two weeks after. The comparison is a correlation for the team to judge.

Keep the number away from individuals. Show each person their own figure and the team's, without names for anyone else, and never rank people or count streaks. The companion guide covers the other mistakes that turn a measure into a target.

TEMPLATE **Leverage goal**

Goal: raise team leverage from $\langle \text{baseline} \rangle \times$ to $\langle \text{target} \rangle \times$ by $\langle \text{date} \rangle$.

Covers: $\langle \text{team, and whose work is included} \rangle$.

Baseline: $\langle \text{value} \rangle \times$, measured over $\langle \text{dates} \rangle$.

Factor to raise: $\langle \text{one factor} \rangle$, from $\langle \text{current value} \rangle$ to $\langle \text{target value} \rangle$.

Changes planned:

1. <change>, owner <name>.
2. <change>, owner <name>.

Read: monthly, on <day>, as a team figure.

TEMPLATE **Change log**

Date	Change to how work runs	Factor it should move	Leverage, 2 weeks before	Leverage, 2 weeks after

12

Where Hamster fits.

A team that runs many agents at once is running a software factory, and Hamster makes that factory build what the team agreed before it builds anything. The team shapes intent together in a Brief, agrees to it by name, and hands it to agents through a Plan. Agents in Claude Code, Cursor, and Codex, and Hamster's own cloud agents, work from the Brief and the team's context and write their decisions back as they work.

You can run everything in this guide from your git host and a spreadsheet. If you want the record to stay current across every person and every agent session, Hamster was built for that.

Common questions

Why is my team still shipping slowly even though every developer uses AI tools?

Agents sped up a part of the work that was not the constraint. Code got cheaper to write, and the hours moved to review, rework, and waiting on people, so the team ships about as much finished work as before.

How do engineering leaders increase team leverage with AI?

Split the team's attention into briefing, steering, review, rework, and reviewing others, and find the largest part. Then raise the factor that shrinks it, one factor at a time, and read leverage monthly as a team figure.

Our AI tools made everyone faster, but now we can't keep up with code reviews. What should we change?

Keep each pull request small and tied to one task in the plan, and review it against the plan. In SmartBear's study at Cisco Systems, reviewers were most effective on 200 to 400 lines at a time.

Is it safe to let AI agents write code overnight without supervision?

Starting overnight runs is cheap. They raise leverage only if the morning review is short and the work does not come back within 30 days of merge.

Do more agents running in parallel make a team more productive?

Only when their work merges without pulling people back in. Parallel agents that each need steering and a long review keep the attention per hour of finished work high, so leverage stays flat.

Why does a feature a developer builds in a day take the team three weeks to ship?

The day of building is fast. The weeks go to agreeing on what to build, rounds of review, and fixes after review, and each of those is attention or a queue in front of a person.

How do you set a goal for engineering leverage?

Record a baseline, set a target in $\times$ and a date, and tie the goal to the one factor you chose to raise. Read it monthly as a team figure, and log every change to how work runs.

Sources

1. Faros AI, The AI Productivity Paradox, 2025. Telemetry from more than 10,000 developers on 1,255 teams; correlational. <https://www.faros.ai/blog/ai-software-engineering>
2. Eliyahu M. Goldratt and Jeff Cox, The Goal. A Process of Ongoing Improvement, North River Press, 1984.
3. SmartBear Software, Best Kept Secrets of Peer Code Review, "Code Review at Cisco Systems," 2006. <https://static0.smartbear.co/support/media/resources/cc/book/code-review-cisco-case-study.pdf>
4. DORA, "Capabilities. Working in small batches." <https://dora.dev/capabilities/working-in-small-batches/>



Run this with your team.

Hamster is free for 10 Briefs a month, with unlimited viewers.

tryhamster.com/get-started

This guide on the web: tryhamster.com/guides/team-leverage