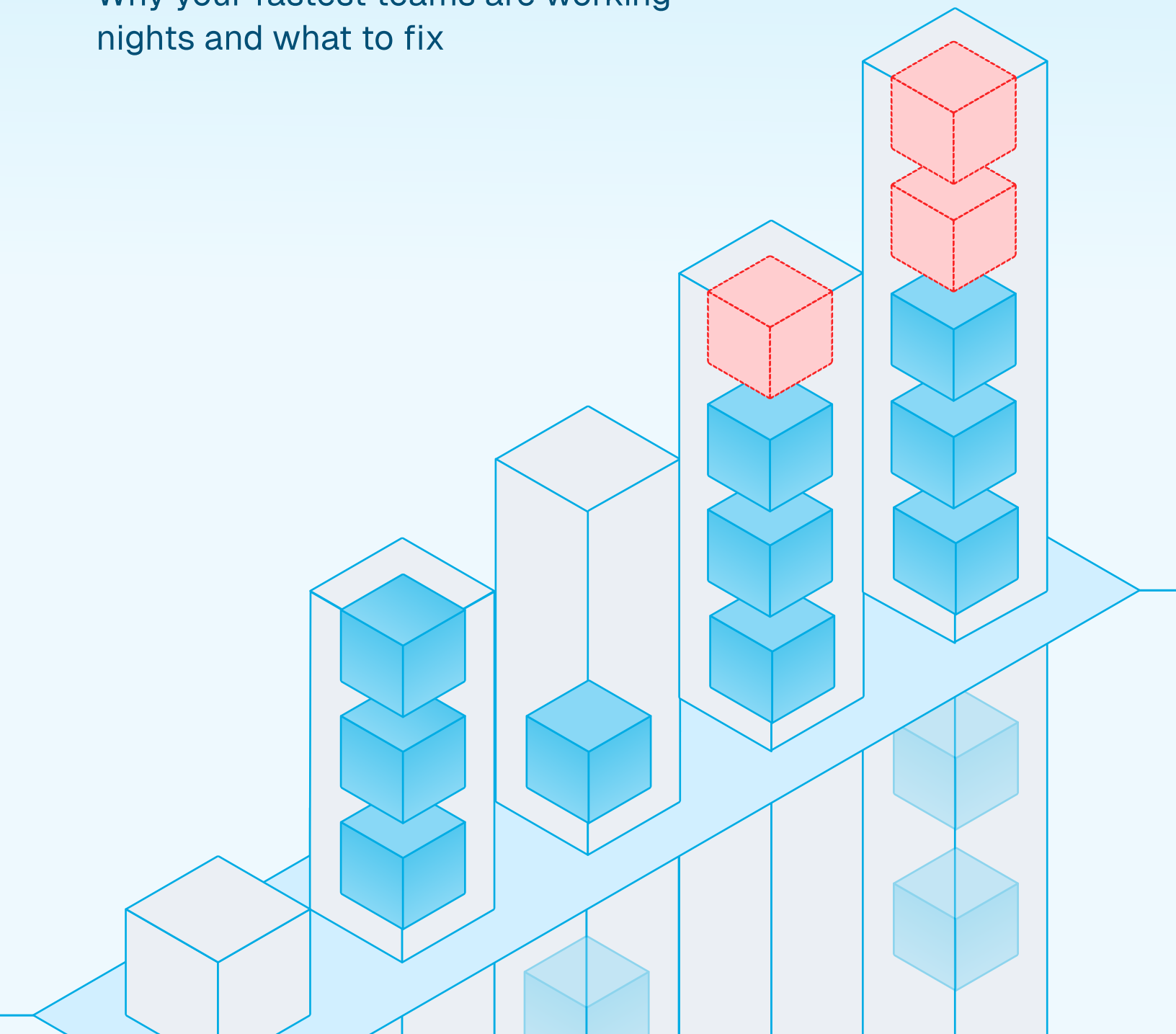




The human cost of AI velocity

Why your fastest teams are working nights and what to fix



Introduction: the other side of the ledger

Our [State of DevOps Modernization](#) report described a system under strain. The teams using AI coding tools most aggressively ship faster than anyone else, and pay for it in more deployment problems, more rollbacks, and slower recovery. That report was about the system: the pipelines, the quality gates, the toolchains. This one is about the people standing inside it.

The numbers are concerning. 52% of developers say they are burned out constantly or frequently. Ask their managers, and 58% say the same about their teams. Burnout at this scale isn't a stressed minority venting. It's the baseline condition of building software in the AI era.

Every rollback is a customer-impacting incident on one side of the ledger and somebody's Tuesday evening on the other.

The strain is downstream of code generation, and so is the relief: the teams that have made their pipelines reliable are the ones whose people have already gone home.

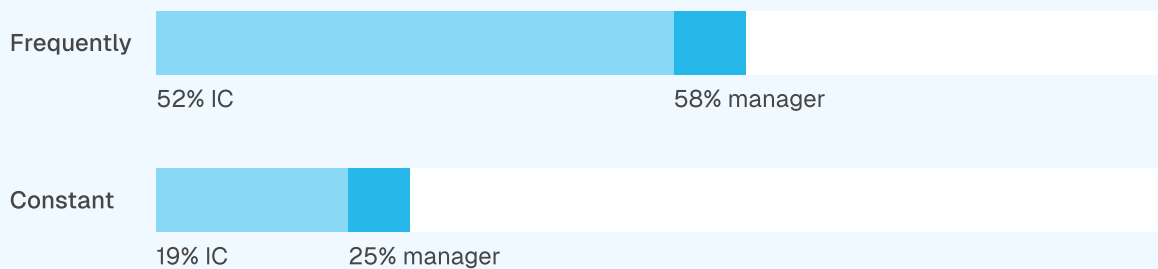
So this report asks two questions. What is actually burning these developers out?
And what does the data suggest we do about it?

Burnout is the baseline

Let's start with the uncomfortable headline. We asked 490 engineering practitioners how often they feel burned out due to work pressures. 52% said constantly or frequently. 19% said constantly. Just 3% said never.

We also asked 210 engineering managers about their teams. Those managers see it too. 58% say developers on their teams show signs of burnout frequently or constantly. A quarter say constantly. Whatever else is true, this is not a perception gap between leadership and the front line. Everyone is looking at the same fire.

Reported burnout with manager stacked on IC reports.



And the fire has a schedule. 31% of respondents say release-related work or production issues pull engineers into evenings or weekends multiple times per week. Another 40% say about once a week. Add it up and 90% are giving up personal time at least a few times a month. Fewer than one in 20 said this rarely or never happens.

In large enterprises, after-hours firefighting isn't a surprise. It's practically a standing meeting.

The reflex is to file this under wellness, schedule a workshop or buy some licenses for a meditation app. But burnout at this scale isn't a personal-resilience problem. It's an operating-model signal. So let's treat it like one and follow the data to its source.

Two sides of the same coin

If late nights and production incidents really are the same problem wearing two hats, the data should show them moving together. It does. Almost perfectly.

Take organizations where more than 30% of code deployments result in a rollback, a hotfix, or a customer-impacting incident. In those organizations, 55% of developers work evenings or weekends multiple times per week, and 55% of managers say their developers constantly show signs of burnout.

Now look at organizations where fewer than 11% of deployments fail. The multiple-late-nights figure drops to 20%. Constant burnout, as observed by managers? Just 4%.

Deployment trouble (rollback, hotfix, or customer-impacting incident rate)	Engineers working evenings/ weekends multiple times per week	Engineers who feel burned out constantly	Managers observing constant burnout in their teams
Under 11%	20%	13%	4%
11-15%	23%	15%	17%
16-20%	30%	17%	14%
21-30%	34%	25%	31%
Over 30%	55%	31%	55%

From 55% to 4%. That's not a modest correlation. We know that when failures are high, we're often asked to release on evenings and weekends for safety. Then, engineers are wrestling with the stress of an outage in the evening when they are already tired. It's a recipe for burnout.

The view from inside the burned-out cohort matches. Engineers who report constant burnout say 23% of their deployments result in a rollback, hotfix, or customer-impacting incident, versus roughly 18% for everyone else. And 93% of them agree their pipelines are "plagued by flaky tests and deployment failures" compared to 39% of developers who rarely feel burned out.

Burnout level	Agree that delivery pipelines are "plagued by flaky tests and deployment failures"	The takeaway here is simple: you cannot fix the business metric and the human metric separately, because they are not separate metrics. Every percentage point of change-failure rate is purchased with somebody's evening. Improve one, and you improve the other. Ignore one, and you ignore both.
Constantly	93%	
Frequently	82%	
Sometimes	58%	
Rarely	39%	

The elite that isn't

We're in a moment that has prioritized speed. By the measure of deployment speed, the cohort shipping to production daily or faster looks elite.

But speed is only half the scorecard. DORA pairs throughput with stability for a reason. Shipping often counts for nothing if much of what you ship breaks, and that's exactly where this cohort comes apart.

Their change-failure rate, measured as the proportion of deployments that end in a rollback, a hotfix, or a customer-impacting incident, averages 25%. For weekly deployers, it's 18%; for monthly deployers, 11%. The faster the cadence, the higher the failure rate, in a clean gradient.

And when something does break, the fastest shipping teams are the slowest to recover: a mean time to restore of 7.6 hours, against 6.3 for weekly deployers. DORA's bar for an elite performer is a change-failure rate of around 5% and recovery in under an hour. This cohort isn't close on either count. This group is fast and fragile.

This trend runs counter to what DORA observed up until 2024, where, for the most part, a lower change failure rate was linked with more frequent deployment. It appeared that reducing risk earned permission to go faster.

The people living inside that trade-off feel it: among daily deployers, 54% work evenings or weekends multiple times per week (versus 22% of weekly deployers and 3% monthly), and 63% report burning out constantly or frequently, against 52% overall. They also lose the most time to toil: 40% of their working hours on repetitive manual CI/CD tasks, against a 36% average. The speed is real. So is the bill.

You can see the same trade from the other direction, and it sharpens the point. The developers who report the most burnout have one defining trait: how hard they lean on AI. 93% use AI coding tools at least daily, 61% multiple times a day. This is more AI-forward than developers who report they suffer from burnout “sometimes” or “rarely.”

Burnout level	Use AI at least daily	Estimated % of deployments that result in hotfix, incident or rollback
Constantly	93%	23%
Frequently	94%	18%
Sometimes	78%	18%
Rarely	67%	19%

They're also the ones mopping up afterward. 74% say AI-generated code leads to deployment problems at least half the time, and 28% say it always does. Heavy AI use, high failure rate, and deep burnout are three things that travel together.

The usual caveat applies: this is correlation, not causation. Demanding, high-stakes environments may drive the velocity, the AI dependence, and the exhaustion all at once. But the shape is consistent across every cut by deploy cadence, by failure rate, and by burnout level and it points in one direction.

Speed bought by skipping stability was never elite performance. It's a loan. The next chapters are about who repays it.

Devs don't blame the robot

Here's where the story takes a turn we frankly didn't expect.

You'd assume the developers telling us AI-generated code causes deployment problems at least half the time would have soured on the technology. They have not.

84% of all respondents agree that "AI tools have reduced my feeling of burnout." And agreement is highest exactly where you'd predict it to be lowest: 88% among the constantly burned out, 91% among daily deployers, and 92% among developers who say AI-generated code always causes deployment problems.

The people putting out the most AI-adjacent fires are the most convinced AI is helping them.

Nor is this a guidance gap. Among the constantly burned out, 93% rate their organization's AI training as good or very good, and 95% agree there are clear expectations for how developers should use AI tools. The inputs are fine. The enthusiasm is real.

And when we asked all 700 respondents what contributes to burnout, the striking thing was how little separated the answers. We offered nine possible sources. Every one of them landed between 67% (unclear priorities) and 75% (manual/repetitive toil).

Burnout contributor	Contributes (sig/somewhat)
Manual/repetitive toil	75%
Pressure to ship quickly	75%
Too many meetings	74%
Security/compliance requirements	73%
Tool switching	70%
On-call/incident load	70%
Slow/unreliable CI/CD	69%
Unplanned work	69%
Unclear priorities	67%

Developers didn't name a single villain.

They endorsed the whole list at nearly the same rate, and among the constantly burned-out every source climbs into the 80s. So why point at pipeline reliability as the lever, when nothing stands out in that ranking? Because a different question separates the burned-out from everyone else, sharply. Asked whether their CI pipelines are "plagued by flaky tests and deployment failures," 93% of the constantly burned-out agreed, against 39% of those who rarely feel burned out. Pipeline pain isn't the loudest complaint on the list. It's the clearest line between the people who are burning out and the people who aren't.

The expectation gap: measured in code, felt in hours

So if the tools and training are good, where is the pressure coming from? The data points to a gap between what gets measured and what gets felt.

Start with the expectations themselves, because they are enormous and they are everywhere. 92% of respondents say AI is now a baseline expectation for modern software delivery, and 82% believe teams that don't adopt AI tools in the next 12 months will fall behind competitors. Almost nobody thinks the gains from AI are modest.

Now compare that to what showed up in the timesheet. We asked how time spent on eight tasks has changed since AI adoption. On seven of the eight, slightly more respondents said time spent has increased than decreased; debugging was the lone exception, and even there the split was essentially even (41% increased versus 43% decreased). Averaged across all eight, increases outweighed decreases, roughly 43% versus 40%. Read that carefully: **after near-universal AI adoption, the workload didn't meaningfully shrink on a single task.**

That's not a contradiction, it's a conversion. The hours AI saved on writing code got spent producing more of it, then reviewing, validating, and debugging the larger volume.

And it isn't a one-survey fluke. Our [State of Engineering Excellence 2026](#) research, a separate study of 700 practitioners and leaders fielded a few months later, found the same pattern from another angle: 81% of engineering leaders say code review time has risen since deploying AI, roughly 31% of a developer's day now goes to AI-related work that shows up in no metric, and the biggest overhead of all, reviewing AI-generated code, is tracked by just 38% of organizations. Output is visible. The validation tax is not.

You can see how the squeeze develops from there. Leadership reads the dashboards as freed capacity; developers experience the same numbers as volume consumed. And the pressure lands hardest on the teams already moving fastest: among daily deployers, 84% say pressure to ship contributes to burnout versus 72% of weekly deployers. The reward for speed, it seems, is being asked for more speed.

Proud, exhausted, and not fooled

There's one more wrinkle, and it echoes the dichotomy between confidence and outcomes from our original report.

The constantly burned-out cohort rates its own DevOps capability higher than anyone else: 93% say good or very good, including 55% who say very good, the highest of any burnout cohort.

The pride is earned. These teams ship daily. If you measure by raw deployment speed, they're winning.

But ask them whether it can last, and the same people give you a very different answer. 84% of the constantly burned out (and 89% of the frequently burned out) agree that "our current ways of working will not be sustainable in the long term."

The people closest to the fire are telling us they're proud of the engine and that it's redlining.

Conclusion: make the system carry the load

The good news is that nobody in this data is asking to slow down. Developers like the speed. They like the AI. What they need is a delivery system that can carry the load their coding tools now generate.

The arithmetic still works. If AI lets your teams produce twice the changes, your pipelines must cut the risk and toil of each change by half or better, or the difference gets paid in human hours and impacts your users.

Fixes are known and well-established:

- Standardized, templated pipelines (golden paths) spread best practices so every new service starts safe.
- Automated deployment verification and automated rollback, so a bad deployment is a non-event instead of an evening of fire fighting.
- Automated security and compliance gates, so the checks keep pace with the code.

AI can also do some of this work directly, if you're deliberate about where. The same signals a person currently reviews by hand, test results, scan output, deployment health, are exactly what AI is good at reading quickly. On a low-risk change like a routine fix, AI can read those signals and clear the change itself, eliminating toil and even a meeting. On a change with real blast radius, a payments system, a regulated workload, a person should still make the final call.

That only works if the system can tell the difference reliably. The same pipeline discipline that makes rollback automatic is what makes it safe to hand a low-risk decision to AI in the first place. This is also where generating and maintaining tests automatically, and building resilience tests from real incident data, pays off twice. They cut toil directly, and they give the automated path a better signal to act on.

One more suggestion, and it costs little: start tracking evenings-and-weekends work as a delivery metric, right alongside deployment frequency and change-failure rate. This isn't an HR metric. It's a core delivery metric. If after-hours work is trending up, your pipeline isn't robust enough.

Build delivery systems that match the speed of AI-assisted development, and the same teams currently burning both ends will simply be what they already almost are: the fastest, proudest engineering organizations in the industry, home by dinner.

Methodology

This analysis is based on a survey of 700 engineering practitioners and their managers from large enterprises, commissioned by Harness and conducted by independent research firm Coleman Parkes in February 2026. The sample included 300 respondents from the United States and 100 each in the U.K., Germany, France, and India.

Notes on limitations

This analysis draws on a single cross-sectional survey. A few limits shape how far its findings can be pushed, and we state them plainly so the numbers are read for what they are.

Point-in-time data. The survey was fielded once, in February 2026. It captures associations at a single moment. It cannot show change over time, and it cannot establish whether AI adoption came before the strain reported here or after it.

Self-reported throughout. Every figure reflects what practitioners and managers told us. Deployment frequency, failure rate, recovery time, and burnout are self-assessments rather than readings from instrumented systems. Perception and measured system behavior can diverge.

Burnout is a single-item measure. Burnout is captured through one self-perception question rather than a validated clinical instrument such as the Maslach Burnout Inventory. The figures describe how often developers say they feel burned out. They should not be read as a clinical prevalence rate.

Means are computed from banded responses. The figures reported as means (change-failure rate, mean time to restore, and share of time on manual toil) are averages built from banded answer options rather than exact values respondents supplied. Open-ended top bands ("more than 30%", "more than 12 hours") were assigned fixed representative values. These means are directional estimates.

Associations without proven causation. The findings are bivariate cross-tabulations, one variable set against another. The analysis does not control for factors such as company size, sector, region, or the underlying intensity of a given environment, any of which could raise both velocity and burnout at once. The same pattern appearing across several independent cuts strengthens the reading. It does not establish cause.

Cohort sizes and definitions. Some sub-groups are small (developers who rarely feel burned out, n=79; monthly deployers, n=32), so any difference resting on them is indicative rather than statistically robust. Burnout cohorts reflect practitioner self-reports, with 476 of 490 practitioners classified. "Daily deployers" is a deployment-frequency grouping from this survey. It is a separate construct from DORA's "elite" tier, which DORA assigns per respondent across all four of its metrics together. Where we compare against DORA benchmarks, we are comparing similar measures that were defined and collected separately.

A companion survey, separately fielded. The State of Engineering Excellence 2026 figures come from a separate Harness-commissioned study of a similar population, fielded roughly three months later. It offers independent corroboration. It uses a different questionnaire and a different respondent set, so the two studies read as consistent with each other rather than identical.



The future of software delivery is autonomous

Harness is the Autonomous SDLC Platform that ships AI code as fast as it's written, catches risk before it ships, and governs every pipeline.



See Harness in Action
harness.io/demo

