Ideals for Developers — Not the Ideal Developer

Marcus Persson Rydberg

March 2026

Contents

The Question in the Pause

A developer is in a job interview. The hiring manager leans forward: “We move fast here. Are you comfortable working nights and weekends when the project demands it?”

The question sounds reasonable. The developer pauses. In the pause, something is known that hasn’t been said.

The question is not assessing competence. It is asking: are you willing to normalize sacrificing yourself for the company’s inability to plan? Are you willing to accept manufactured urgency as the default operating mode and call it passion? The question is a filter — and what it filters for is compliance with a system that externalizes its costs onto developers, and onto future codebases that no one in this room will maintain.

What the developer wants to say, and almost never does: “I’ll do what it takes in a true emergency. I won’t normalize treating every week as an emergency. Those are different things, and the distinction matters.”

The role being offered is not for the developer who does great work. It is for a person-shaped tool.

This essay is about the distinction that surfaces in that pause. The claim is simple: being happy, relaxed, and authentic produces better code and better teams. This is not a claim about working less. It is a claim about which internal conditions produce the judgment, taste, and verification that software now requires most. The AI era makes this argument more urgent, not less.

The State Being Defended

Before the evidence, a definition. The word “relaxed” is easily misread as passive or unambitious. The state being defended here is something more specific.

There is a quality of mind that produces the best software, and it is not the quality produced by fear. It is not relaxed in the sense of low stakes. The developer in this state may care intensely. They may stay late because the problem still has them. But there is a difference — felt in the body, visible in the code — between staying because the problem has you and staying because someone will think less of you if you leave.

What this state actually consists of: the freedom to think out loud without editing the thought before it forms. The ability to say “I don’t know” without that admission becoming a permanent mark in someone’s mental ledger. The willingness to pursue a direction that might be wrong, because being willing to be wrong is the

only way to find what's right. The capacity to sit with an unresolved problem long enough that the problem itself tells you what it needs.

None of this looks productive from outside. It doesn't show up in commit graphs. It is what happens in the hour before the commit, in the quiet between the PRs, in the walk that looks like avoidance but is actually where the work gets done. Environments that optimize for visible productivity destroy this state because they make the invisible work dangerous.

The ideal being defended is also not a call for comfort over craft. A developer who is happy is not one who is never challenged — they are one who finds the work meaningful. A developer who is authentic is not one who avoids conflict — they are one whose work reflects genuine values rather than a performed role. These ideals are in direct tension with what might be called *struggle-porn culture*: companies that brag about intense hours, treat exhaustion as a signal of commitment, and measure work by presence rather than output.

I. What We Know

Psychological safety — the strongest single finding

Google's Project Aristotle studied over 180 teams and found that psychological safety — the shared belief that the team is safe for interpersonal risk-taking — was the top predictor of team effectiveness, ranking above individual skill, seniority, or team composition. Amy Edmondson (Harvard Business School) first operationalized the concept in 1999 in hospital teams. Her mechanism: psychological safety enables learning behaviors — questioning, admitting uncertainty, reporting failures, proposing unproven ideas — that are necessary for complex, knowledge-intensive work. Under threat, people protect themselves. Under safety, they improve the system.

The direct application to software is not indirect. A team where developers fear judgment for imperfect PRs, wrong estimates, or "I don't understand this" conversations will conceal uncertainty, ship code they're not confident in, and miss the early architectural problems that are cheap to fix and expensive to defer. This is not a culture problem with soft consequences — it is an engineering problem with hard ones.

The DORA finding: elite teams don't trade off

The DORA research (DevOps Research and Assessment, annual since 2014) is the largest ongoing empirical study of software delivery performance. Its central finding, replicated across nine years: elite-performing teams are also the ones with generative cultures, psychological safety, and lower developer burnout. Speed and stability are not in tension at the elite level — they are correlated. The teams deploying most frequently also have the lowest change failure rates.

This result directly defeats the "you can have high performance or you can have wellbeing, not both" argument that underlies struggle-porn culture. The DORA data shows that culture predicts delivery performance more strongly than any single technical practice.

The cultural underpinning is Ron Westrum's typology (2004): organizations can be pathological (power-oriented, messengers shot, failure punished), bureaucratic (rule-oriented, messengers tolerated), or generative (performance-oriented, messengers rewarded, failure treated as learning). Software teams in the generative typology outperform on every delivery metric.

SPACE and the developer satisfaction argument

The SPACE framework (Forsgren, Storey et al., Microsoft Research / GitHub, 2021) measures developer productivity across five dimensions: Satisfaction and wellbeing, Performance, Activity, Communication and collaboration, Efficiency and flow. The first dimension is not an afterthought — the authors explicitly argue that developer satisfaction is a productivity *input*, not just a soft outcome.

The most important structural insight from SPACE: over-optimizing for Activity (lines of code, commits, tickets closed) can actively degrade Satisfaction and Communication. The framework reveals the metric-corruption dynamic in measurable form: the proxy destroys the thing it was supposed to track.

Mood and code quality

Graziotin et al. (IEEE Transactions on Software Engineering, ~2017) is the most direct individual-level evidence: developer affect correlates with problem-solving quality and bug rates. Positive developer mood correlates with better problem-solving performance and fewer bugs; negative affect with disengagement and higher error rates. This is not team-level data, which strengthens the mechanistic argument — it runs through individuals, not just cultural aggregates.

The cost of interruption

Gloria Mark's research at UC Irvine finds that after an interruption, average recovery time to return to the prior cognitive state is approximately 23 minutes. Interrupted workers compensate by working faster but make more errors and report higher stress. Parnin and DeLine (ICSE, 2010) extend this to software specifically: developers must reload not just the task but the state — where they were, what they were thinking, what the next step was.

A day with six 30-minute meetings does not cost three hours. It prevents deep work entirely.

The crunch counter-experiment

The game industry is the most documented natural experiment on sustained overwork in software development. The evidence is consistent across post-mortems, IGDA developer satisfaction surveys, and detailed accounts (Jason Schreier's *Blood, Sweat, and Pixels* and *Press Reset*): crunch increases the rate of errors while simultaneously reducing the team's capacity to catch them. Both the error rate and the review quality are affected by exhaustion. The practice persists not because the evidence supports it but because the culture that produces it also produces the framing that normalizes it.

II. The Startup Case

The counter-argument must be stated fairly before it can be answered. The strongest version: most startups are *default dead* — burn rate exceeds path to sustainability. The only escape is growth fast enough to change the fundraising timeline. In winner-take-most markets, second place may be worth nothing. This creates genuine structural pressure for speed that looks like superhuman effort from outside. Paul Graham's essays frame this as physics: increase the number of trials per unit time to raise the probability of finding product-market fit before the money runs out.

This argument applies most cleanly to the early-startup survival phase. Concede the domain.

What the evidence does not support is the extension of this frame beyond the survival phase — and the evidence on what the survival phase actually requires is less favorable to intensity than is typically claimed. CB Insights, Startup Genome, and Failory analyses of startup failures consistently show the primary causes as: no market need, ran out of cash, wrong team composition, getting outcompeted, and product problems.

Culture factors appear in roughly 5–10% of cases, typically as toxic team dynamics (fear, conflict, blame), not as insufficient intensity. If chronic intensity were causally necessary for success, "insufficient drive" would appear regularly as a failure cause. It does not.

The correlation between intense startup cultures and success is largely selection. Intense cultures attract people already predisposed to high-intensity work. Those who aren't fit leave. The observed high output may be mostly a function of who passed the filter rather than what the culture produced from a general population. Repeat founder testimony is consistent: first-venture intensity is typically emergent from inexperience and genuine mission alignment, not something that can be manufactured by management and applied to others.

The humane-growth-at-scale counter-examples also matter. GitLab reached a $6.5B IPO with a fully remote, async culture and explicit "no hero culture" values. Atlassian's co-CEOs stated their opposition to all-nighter culture publicly before the company completed its IPO — not after success was secured. Mailchimp was acquired for $700M bootstrapped, with a documented anti-hustle stance.

The honest synthesis: intense effort from people who are intrinsically motivated about a specific problem can produce extraordinary output. When companies apply this as managed policy — extracting intensity from people who don't share the intrinsic motivation through social pressure, surveillance, and fear — they get the cost without the output.

There is also a distinction that rarely gets named in startup culture discourse, between genuine and manufactured urgency. *Genuine urgency* is external in origin, time-bounded, shared, and rare. *Manufactured urgency* is internal, has no resolution state, and is the default operating mode. Real urgency burns hot and clean. Manufactured urgency leaves ash. The distinction is not detectable from a commit graph.

III. The AI Era

A 2022 GitHub/Microsoft study found developers using Copilot completed a well-defined task 55% faster than controls. The figure is cited everywhere. What it actually measures — a single, narrow, JavaScript task, with no follow-up on cognitive load, code quality at scale, or wellbeing over time — is cited nowhere near as often. The complementary finding from Sandoval et al. (2022): Copilot-assisted developers produced code with significantly more security vulnerabilities, and were often unaware they had done so. The "faster" and "less secure" findings are both real. Together they tell a more honest story: AI tools accelerate generation

while the verification burden they create is real and consequential.

As generation becomes cheap, the bottleneck shifts. The expensive operations are now: architectural judgment (what to build, how it fits, what the second-order consequences are); taste (what is the right solution, not just a working solution); verification (is this AI output correct, secure, and idiomatic in ways that matter); and problem decomposition (framing the query that gets useful output). These are not rote tasks. They are high-cognitive-load, experience-dependent, context-saturated operations that require exactly the mental state that sustained stress degrades.

This means the "wellbeing is nice but the company needs output" framing misunderstands which output now matters. A developer who produces twice as many AI-generated lines under pressure but whose architectural judgment is impaired produces compounding technical debt, security vulnerabilities, and wrong-direction work at a cost that exceeds any line-count gain.

There is also a risk in the opposite direction: AI-powered productivity surveillance. The tools exist — commit frequency analysis, PR cycle time tracking, code review rate monitoring — and the incentive to apply them as performance management tools exists too. Organizations using AI-derived productivity metrics to raise throughput expectations are extrapolating far beyond what the evidence supports. In adjacent knowledge-work fields the pattern is documented: AI monitors throughput, management uses the data to set new baselines, workers face escalating targets with no natural ceiling. The results are documented increases in stress, monitoring anxiety, and reduced job satisfaction. The monitoring system creates a specific rational incentive: approve AI-generated code quickly (visible throughput) rather than perform the slow scrutiny that catches subtle errors (invisible, penalized by cycle time metrics).

A third dimension of the AI era argument is worth naming honestly: the emotional experience of watching a significant fraction of your technical fluency become cheap. Not obsolete — that is the wrong word. The skills still work. Their rarity has changed. There is a difference between a thing having no value and a thing having stopped being a differentiator, and the emotional register of the two is completely different. The second is a kind of grief that doesn't have a name yet.

What remains when the generation is cheap is harder to hold and more important: the ability to look at the AI's output and know what it doesn't know it doesn't know. The unstated case, the production environment assumption, the edge at the intersection of two behaviors the test suite doesn't cover. This was always the most valuable part of the work. AI makes it visible by removing everything around it. The honest emotional experience is grief mixed with liberation, and both are real simultaneously.

IV. The Mechanism

For a technical audience, the case should not rest on correlation alone. There is a mechanistic chain that connects the neuroscience of stress to the operations that now constitute the bottleneck in software work.

Link one: stress impairs prefrontal cortex function. Arnsten (2009, *Nature Reviews Neuroscience*) established the mechanism: cortisol and norepinephrine, released under stress, activate the amygdala and simultaneously suppress prefrontal cortex function. The PFC governs working memory (holding multiple considerations active simultaneously), cognitive flexibility (switching between framings, updating understanding), and inhibitory control (suppressing the first plausible answer in order to reach a better one).

The Yerkes-Dodson inverted-U describes the relationship between arousal and performance: performance rises with moderate arousal and falls with high or sustained arousal. Crucially, the peak depends on task complexity. Simple, well-defined tasks have a higher peak. Complex, ambiguous tasks requiring integration have a lower peak and a steeper falloff. The tasks being displaced by AI are exactly the ones where the Yerkes-Dodson argument for pressure is strongest. The tasks that remain are exactly the ones where it is weakest.

McEwen (1998, *New England Journal of Medicine*) established the duration dimension: allostatic load is the cumulative physiological cost of chronic or repeated stress activation. Acute stress followed by genuine recovery accumulates little allostatic load. Chronic stress without recovery accumulates structural cost — not just fatigue, but degradation of the PFC itself.

Link two: interruptions fragment the cognitive work that matters. Mark, Gudith & Klocke (CHI, 2008) found that knowledge workers take an average of 23 minutes to return to a task after an interruption. Parnin and DeLine (ICSE, 2010) extend this to software specifically: developers must reload not just the task but the state — where they were, what they were thinking, what the next step was. In AI-assisted development, the verification task that now constitutes the core of the work is precisely the task most disrupted by interrupted attention.

Link three: repetitive high-stakes judgment fatigues. Danziger, Levav and Avnaim-Pesso (PNAS, 2011) documented Israeli parole judges' favorable ruling rates falling from approximately 65% at the start of a session to nearly zero just before a break, then recovering fully after the break. The analogy to AI code

verification is structural: both involve sequential scrutiny of discrete items, high stakes if errors are missed, no external signal reliably identifying which items require more scrutiny, and no natural endpoint. The tendency under fatigue is to default to the less effortful response. For developers reviewing AI-generated code, the fatigued default is approval — accepting code that looks roughly right without the scrutiny that would catch the subtle error.

Link four: recovery is structurally required. Harrison and Horne (2000) documented a consistent pattern in sleep deprivation studies: routine, well-practiced task performance degrades less under sleep loss than innovative, flexible, plan-updating decision-making. The capacities most sensitive to sleep restriction are exactly the ones the argument identifies as the AI-era bottleneck.

The mechanism also runs through surveillance. Amabile (1996) showed that expected evaluation degrades creative performance by approximately 20% — specifically impairing open-ended, integrative, exploratory thinking while leaving rote task performance relatively intact. Monitoring on commit frequency and PR cycle time rewards high-velocity visible activity. It penalizes the deep, slow work that architectural judgment and careful code review require — which looks like low activity from outside.

V. The Code as Evidence

The argument so far is empirical and mechanistic. There is also a kind of evidence that doesn't appear in any study — but that engineers with enough experience recognize immediately.

A codebase reflects its culture.

Code written under chronic pressure has a characteristic signature: it handles the case the developer was thinking about when they wrote it, and not the others. The edge cases are missing — not because the developer didn't know about them, but because the cognitive state that anticipates what you don't know requires a kind of spaciousness that pressure forecloses.

Read the diffs. You can often tell which code was written in flow and which was written under duress. One is tight, confident, occasionally a minimal abstraction that does more than it looks like. The other is verbose and defensive, its escape hatches visible everywhere — the proliferation of `TODO` comments, the special-cased branches, the function that grew past its original purpose because there wasn't time to refactor and the developer knew it but kept going anyway.

Two kinds of fatigue

There are two kinds of tired after a hard day of writing code.

One is clean, like the ache from a long hike. The problem was genuinely difficult and you were genuinely inside it. You followed a trail of logic until it opened onto something. You forgot to eat. Time collapsed. The code you produced in those hours has a certain quality — not because you were trying to be elegant, but because you had the uninterrupted time to find the core shape of the thing. You are tired in the way that means something was used well.

The other fatigue is different. It's dirty, sticky. You worked as many hours and produced as many lines, but the work was performed rather than done. You were thinking about being seen working rather than thinking about the problem. The function you wrote handles the case that was specified. You knew while you wrote it that there were other cases, but checking would mean slowing down, and slowing down would look like not trying. The fatigue here is the fatigue of alienation — of having been somewhere else while your hands moved.

Developers know these two states in their bodies. The distinction almost never comes up in any review, retrospective, or performance conversation. The cultures built around the second state call it productivity.

The Carmack distinction from the inside

John Carmack working through the night because the renderer glitch had teeth — because solving it was the puzzle pulling him forward — is a different thing from a developer grinding past midnight under the ambient pressure of a culture that calls exhaustion dedication.

From outside, they look the same. Both are still at the keyboard at 2am. But the phenomenology is entirely different.

In the first state, the pull feels like flying. Time dissolves. Coffee goes cold. The breakthrough arrives with a quality of surprise even when you were almost there. The code that comes out has contact with the material — it isn't just correct, it shows that someone was actually thinking.

In the second state, the push feels like wading through mud. You can see the next step but not two steps ahead, and the architecture of what you're building is increasingly opaque to you. The code that comes out is done. Rarely delightful. It handles the stated cases. The developer knew while writing it that it was B-work. There was no other available gear.

What Carmack's example actually demonstrates is that the conditions for extreme productive effort are internal. They cannot be manufactured by someone else. Trying to replicate the output without the internal state produces something that looks similar on a time-tracking spreadsheet and is categorically different in quality. The companies that invoke Carmack as justification for demanding intensity have misread the example in a way that is convenient for them.

The ideals this thesis defends do not produce passivity. They produce a specific kind of attention: to what matters and what doesn't, to genuine urgency when it arrives, to the difference between a problem that requires slowing down and a problem that requires moving. The relaxed state is not low-arousal disengagement. It is the absence of threat, which is the precondition for the kind of presence that complex work requires.

Appendix: What Organizations Have Built

The following cases document what specific organizations have built and what they say about why. This appendix does not tell anyone what to do. The reader draws the inferences.

None of these cases constitute proof that these practices caused their success. The evidence for causation is not present. What they document is that organizations at significant scale, in competitive markets, have explicitly rejected intensity-as-virtue as a permanent operating mode — and described their reasoning.

GitLab

Primary source: handbook.gitlab.com (~2,000 pages, publicly versioned)

> "We don't want heroics. We want sustainable processes. If you find yourself in a situation that seems to require heroics, that's a signal of a process problem, not an individual problem."

> "Working more than you should is a sign of a process problem, not a virtue."

GitLab's async-first communication norm emerged from a distributed team across timezones. The handbook's stated reasoning is operational: written communication produces a permanent record and preserves meeting time for decisions that require it. The "no hero culture" value is explicitly framed as a systems argument: heroics create single points of failure and mask underlying fragility.

What this documents: Heroic effort was classified as a process-failure signal — not a cultural value — in a publicly versioned handbook at a company that completed a $6.5B IPO.

Basecamp / 37signals

Primary sources: It Doesn't Have to Be Crazy at Work (Fried & Heinemeier Hansson, 2018); Shape Up (public)

> "Calm is what you get when you stop chasing 'crazy.' Calm is collected, considered, and in control… The most brilliant things are done when you're at your best, not your worst."

> "If you can't complete your work in a normal week, we want to understand why, and fix the problem, not make the person work more."

Fried and Heinemeier Hansson state explicitly that their model works for them and that they are not writing a universal prescription. Their independence and size give them options that larger, investor-driven companies may not have. Shape Up's six-week cycles with built-in cool-down periods between cycles are the most operational version of the recovery argument: recovery is structured into the work rhythm, not left to individual discretion.

What this documents: Sustained calm and protected maker time were deliberate operational choices articulated at length over 20+ years, with trade-offs acknowledged in the same text.

Shopify

Primary sources: Tobi Lütke, NYT 2016; Invest Like the Best podcast 2019; shopify.engineering

> "Psychological safety is the most important thing in a team. If people can't say 'I think this is wrong' or 'I don't understand this' without fear, you're going to make worse decisions." — Tobi Lütke

> "The person who's going to spot the problem is often the most junior person in the room, because they're not carrying the assumption that the thing is going to work. If they can't speak, you lose that signal." — Tobi Lütke

Shopify's "trust battery" is an operational metaphor used internally: every relationship starts at 50% charge, and actions either charge or drain it. Psychological safety is framed not as a wellness initiative but as a decision quality mechanism. Suppressing uncertainty suppresses the signal that catches errors.

What this documents: Psychological safety was framed as an operational decision-quality tool during the company's growth phase, not retrofitted after scale.

Atlassian

Primary sources: Scott Farquhar, AFR 2019; Mike Cannon-Brookes interviews; TEAM Anywhere documentation

> "I don't believe in the all-nighter culture. I've never pulled an all-nighter at Atlassian and I'm proud of that." — Scott Farquhar

> "We've always tried to build a company where people don't have to work ridiculous hours. We think you do better work when you're rested." — Mike Cannon-Brookes

Both co-founders stated this position publicly before Atlassian completed its IPO — not after success was secured. This removes the hypothesis that sustainable culture is a post-success luxury adopted only once the hard part is over.

What this documents: The anti-crunch position was on record from both founders before scale, under competitive pressure, in a market where intensity-culture competitors existed.

Valve

Primary source: Valve Employee Handbook (confirmed authentic by Valve, leaked 2012)

> "We want innovators, and that means maintaining an environment where they'll flourish… The reason is that the best people want to be self-directed."

The Valve handbook is unusual for the candor of its reasoning. It explains not just what Valve does but why — including acknowledged costs. The handbook explicitly notes that radical self-direction "can feel terrifying when you first start. We know." Gabe Newell has acknowledged that the model produces costly hiring mistakes and can create coordination failures.

What this documents: Radical self-direction was a deliberate design choice based on an explicit theory of human motivation and creative output, with the founders acknowledging both benefits and costs in the same document.

References

- Forsgren, Humble, Kim — *Accelerate* (2018)

- Edmondson — *The Fearless Organization* (2018)
- Graziotin et al. — IEEE Transactions on Software Engineering (~2017)
- Arnsten — *Nature Reviews Neuroscience* (2009): "Stress signalling pathways that impair prefrontal cortex structure and function"
- McEwen — *New England Journal of Medicine* (1998): "Protective and damaging effects of stress mediators"
- Mark, Gudith & Klocke — CHI (2008): "The cost of interrupted work"
- Parnin & DeLine — ICSE (2010): "Resuming programming tasks with execution backtracking"
- Danziger, Levav & Avnaim-Pesso — PNAS (2011): "Extraneous factors in judicial decisions"
- Amabile — *Academy of Management Journal* (1996): "Creativity and innovation in organizations"
- Westrum — *Quality & Safety in Health Care* (2004): "A typology of organisational cultures"
- Forsgren, Storey et al. — *Queue* (2021): "The SPACE of Developer Productivity"
- Sandoval et al. (2022): "Do Users Write More Insecure Code with AI Assistants?"
- Sennett — *The Craftsman* (2008)
- Newport — *Deep Work* (2016); *Slow Productivity* (2024)
- Schreier — *Blood, Sweat, and Pixels* (2017); *Press Reset* (2021)
- Kellogg, Valentine & Christin — *Academy of Management Annals* (2020): "Algorithms at Work: The New Contested Terrain of Control"
- Walker — *Why We Sleep* (2017)
- Weick & Sutcliffe — *Managing the Unexpected* (2001/2007)