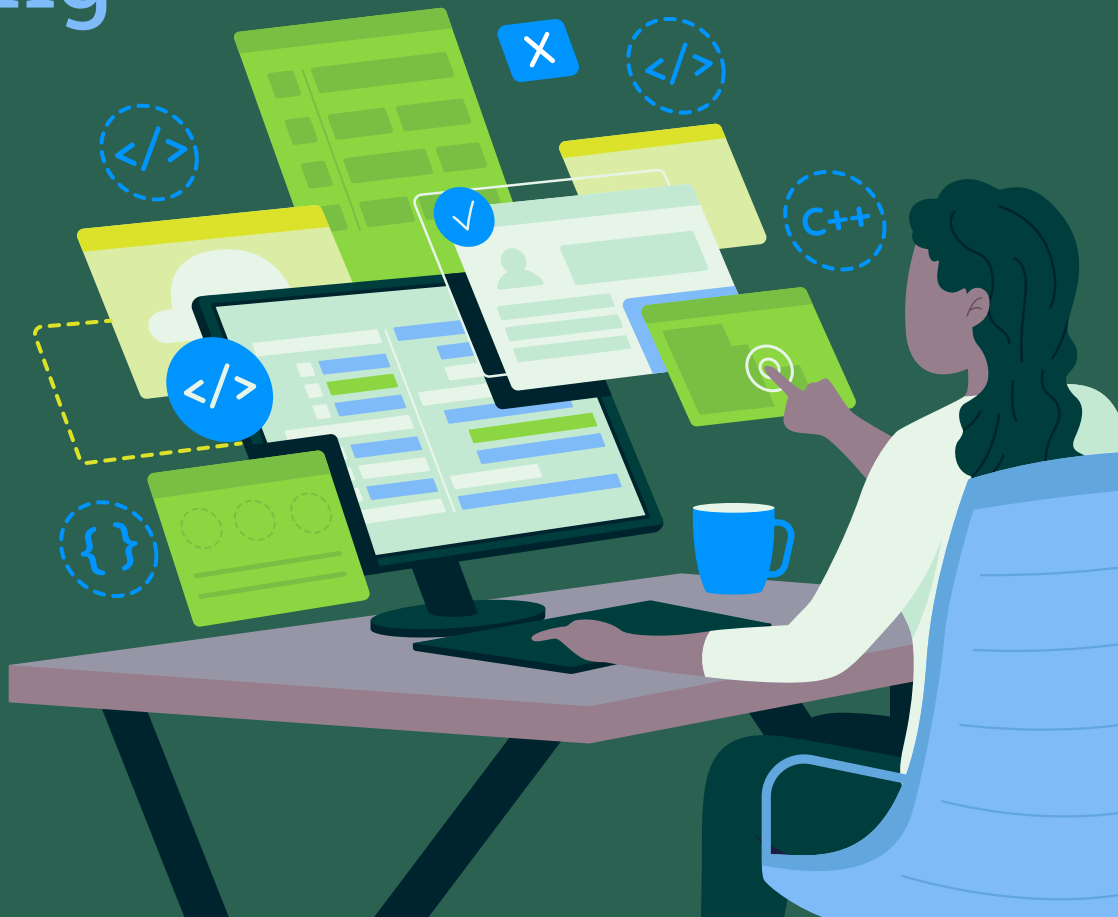
THE LINUX FOUNDATION | Research

DEVREL FOUNDATION

Optimizing Developer Experience for High Engineering Performance

A Roadmap for Enterprises to Reduce Friction, Measure Impact, and Scale Productivity in Software Delivery



September 2026

Ibrahim Haddad, Ph.D., *Volvo Cars*

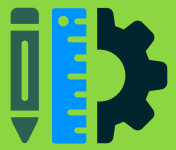
Foreword by Arun Gupta, *NVIDIA*

Optimizing Developer Experience for High Engineering Performance

Most organizations send their engineers out to run a marathon with no markers on the course, then call the burnout discipline. **Developer experience is the markers.**

Developers lose more than 17 hours every week to maintenance and bad code, costing roughly $300 billion in global GDP each year.

69% of developers lose at least eight hours a week to inefficiencies, with technical debt and bad documentation leading the list.



Best-in-class developer tools are the single largest driver of software-related business performance, ahead of culture, product management, and talent.

Treat developer experience as a delivery KPI, not a perk, and quality, speed, and retention follow.

If you only track four numbers, track lead time for changes, deployment frequency, mean time to recovery, and change failure rate.

Cognitive load is a design problem, not a wellness problem.

The developer environment deserves the same treatment as production infrastructure: fast, integrated, stable.

More frustration produces less output, less output justifies more pressure, and more pressure pushes more people out.

Improving DevEx rests on three pillars: streamlining workflows, investing in tooling, and building an engineering culture worth staying in.

When developers have real authority over technical decisions, they move faster, experiment more, and own the outcomes.

Outcome metrics should only be used to evaluate DevEx initiatives after sufficient time has been allowed for the environmental improvements to take hold and begin producing results.

Contents

Foreword

You can send someone out to run a marathon, but without mile markers or a pace clock, even a strong runner will burn out by mile six or coast so easily they leave time on the table. The distance doesn't change. What changes is whether the runner can feel where they actually are in the race, instead of trusting whatever their adrenaline or nerves are telling them. That is what developer experience is, underneath all the metrics and frameworks in this report: not making engineers run faster, but giving them the markers to know how they're actually doing while they run.

I have spent over two decades moving between companies that make software for a living across Sun Microsystems, Intel, Amazon, Apple, and now NVIDIA, and the one thing that never changes is how much a team's daily friction shows up months later in the product. You can tell, walking into an engineering organization, whether the build takes ten seconds or ten minutes, whether a new hire can see how far along they are in week one or is running with no markers at all, whether the person next to you trusts the documentation or has learned the hard way to ignore it. None of that shows up on a roadmap slide. All of it shows up somewhere around mile eighteen, right when you hit the wall.

That is what makes this report useful rather than aspirational. Ibrahim Haddad has spent his career on the side of engineering that gets built rather than talked about, and it shows in how carefully this report separates what sounds good from what actually tells a runner where they stand. The instinct in most organizations is to treat developer experience as a wellness initiative, something you fund after the roadmap is safe, not something the roadmap depends on. Building DevEx in from the start doesn't make anyone run faster on the first mile. It's more like clearing lactic acid before it builds up, so the runner isn't paying interest on miles one through ten by mile eleven. This report makes the case that most organizations are running their engineers through a marathon with no markers on the course at all, and calling the burnout discipline. The $300 billion figure from Stripe's research alone should be enough to move DevEx out of the "nice to have" column in any budget conversation.

As I build open source practices at NVIDIA, I spend a lot of my time thinking about how to give our own developers that same kind of mile marker, in the form of clear, checkable open source practices, rather than leaving them to guess where they stand. I built a small tool because I was tired of guessing whether a repository was healthy the way you'd guess how a runner is doing just by watching their face, instead of checking a watch. Reading this report's case for trendlines over thresholds, leading indicators over lagging scorecards, felt like someone had put an actual pace clock on an argument I had been making by feel.

If you run engineering teams and have never measured any of this, start with the four DORA metrics in Table 2, your mile markers. Most organizations still send their best engineers out to run with no markers on the course. This report is a good place to start setting a few.

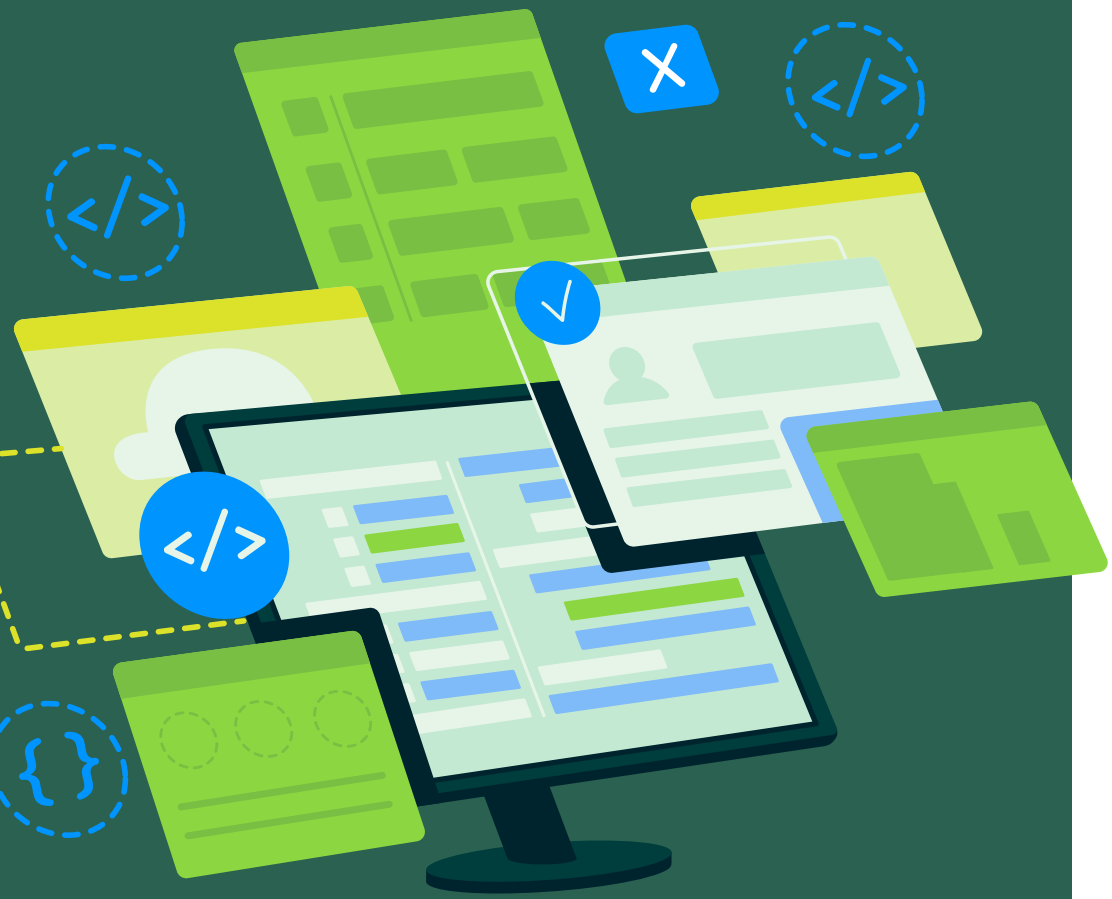
Arun Gupta
Director, Open Source, NVIDIA

Executive summary

Enterprises must keep optimizing how they build software if they want to stay agile and efficient. One factor often gets overlooked: Developer Experience (DevEx), the quality of the interactions developers have with their tools, processes, and work environment. DevEx has a direct effect on engineering output, product quality, and business results.[1, 2, 3, 4] Poor DevEx drives friction, lengthens cycles, and pushes good engineers out the door.

Enterprise software development brings its own obstacles. Bureaucratic or lengthy approval cycles, fragmented tooling, inefficient workflows, and high cognitive load slow teams down and erode morale. Enterprises looking to fix this and improve their DevEx require a structured approach: better tooling, cleaner workflows, and a culture that backs engineering work.

This report sets out the methodologies and actions enterprises can take to improve DevEx, along with the metrics that show whether those changes are working. When enterprises invest in DevEx, they create a more efficient, motivated, and high-performing engineering organization, which in turn produces better software and better business outcomes.



The strategic importance of DevEx

Enterprises are under constant pressure to ship high-quality software while keeping the innovation pipeline full. Adopting newer technologies or rolling out agile practices is not enough on its own. The experience of the developers themselves, how they interact with their tools, workflows, and the wider organization, has a direct effect on output and business performance.[3, 4]

Developer Experience covers what developers actually deal with: their tools, the processes around them, and the engineering culture they sit inside. When organizations get it right, development cycles shorten, collaboration improves, and engineers can focus on real problems. When they get it wrong, the cost is visible through slower delivery, brittle workflows, and engineers updating their LinkedIn profiles.

DevEx has historically sat in the blind spot of enterprise software strategy. Many organizations focus on optimizing isolated processes or buying new technology without examining what the day-to-day actually looks like for the people writing the code. Engineers end up navigating friction points that drag down output and disengage them from the work.

This report gives enterprises a structured approach to improving DevEx: identifying the key challenges, recommending practical strategies, and establishing measurable metrics to track progress. The sections that follow break down the core components of DevEx, the barriers that hold developer productivity back, and the frameworks enterprises can adopt to make real improvements.

Understanding DevEx

DevEx is how developers experience their tools, workflows, and the organization around them. One clarification up front, because the term travels: in developer relations, developer experience often means the product-facing experience external developers have with an API, SDK, or documentation. This report uses it in the internal sense, the experience an enterprise's own engineers have building software inside the organization. It shapes productivity, job satisfaction, and the team's ability to ship quality software at pace. DevEx is the lived day-to-day of writing, reviewing, debugging, and deploying code. In enterprise environments, where development is large-scale and complex, DevEx is a strategic concern.

- The quality of DevEx has direct, measurable effects[4]:
- Faster development cycles: Less time fighting friction means more time delivering value.
- Higher code quality: Well-supported teams produce more reliable software with fewer defects.
- Better retention: Engineers stay where the work environment respects their time and craft.
- Stronger business outcomes: Faster, higher-quality delivery keeps enterprises competitive.

With these foundations in place, the next sections look at what DevEx is built from, where it tends to break down, and how to fix it.

Key elements of DevEx

DevEx breaks down into five core elements: tooling and infrastructure, processes and workflows, cognitive load, collaboration and communication, and organizational support. Each shapes a different part of the day-to-day developer experience. This grouping aligns with the actionable DevEx framework proposed by Greiler, Storey, and Noda, which organizes friction sources across human, technical, and organizational dimensions.[5]

Tooling and infrastructure

Developers work daily with IDEs, version control, build automation, and CI/CD pipelines. When these tools are slow, unreliable, or poorly connected, every action takes longer, and frustration builds. High-performing teams invest in well-integrated, fast, stable environments that minimize disruption.

Processes and workflows

In large enterprises, software development often runs through long approval chains, heavy compliance requirements, and inefficient handoffs. Rigid, non-developer-friendly processes turn into bottlenecks. Trimming unnecessary steps, automating repetitive tasks, and adopting agile practices removes much of that drag.

Cognitive load and developer focus

Developers need uninterrupted time to write and debug code. Constant context switching, too many meetings, and unclear requirements raise cognitive load and lower output. The goal is an environment where engineers can think deeply about problems instead of navigating organizational complexity. Skelton and Pais argue that team cognitive load should be treated as a primary design constraint, not an afterthought.[6]

Collaboration and communication

Effective communication between developers, product managers, release managers, designers, and other stakeholders is essential. Poor documentation, vague requirements, and misaligned expectations create rework and slow everything down. Enterprises need to build a culture of clear communication for developers to have the context and resources to do their work well.

Organizational support and engineering culture

A strong engineering culture takes developer feedback seriously, supports continuous learning, and treats well-being as a real concern, not a slogan. Leadership shapes this by funding developer support systems, mentorship programs, and career growth opportunities.

Challenges hindering DevEx in enterprises

Many enterprises understand that DevEx matters, yet struggle to act on it. Structural inefficiencies, legacy constraints, and misaligned priorities make it hard to optimize the developer journey. The result is slower software delivery, frustrated engineers, and rising attrition. Three obstacles stand out: organizational and process barriers, fragmented tooling, and the toll of cognitive load on developers. Addressing them across an organization takes coordinated work across operations, environments, and culture.

Organizational and process barriers

Enterprises often run on rigid structures that put governance and risk management ahead of agility. Bureaucratic approval chains, slow decisions, and meeting overload pile friction onto the development lifecycle. Engineers spend too much time navigating internal processes and too little time writing and shipping code.

Communication gaps make this worse. When engineering, product, and business teams pull in different directions, requirements stay vague and last-minute changes derail work in progress. Excessive meetings cut into the deep, focused time developers need to be effective.

Fixing organizational barriers means moving toward leaner, more developer-friendly processes: fewer approval steps, more asynchronous communication, and real decision-making authority for the teams closest to the work. Even with these changes, developers still face friction from fragmented tooling and workflows.

Fragmented tooling and inefficient workflows

In many enterprises, teams work with an inconsistent mix of tools, frameworks, and environments. Without standardization, work gets duplicated, compatibility issues stack up, and switching between projects becomes its own learning curve. Outdated tools and poorly connected development environments slow workflows instead of supporting them.

CI/CD pipelines, central to modern delivery, are often patchwork or under-optimized. When developers spend hours debugging build failures, working around broken dependencies, or waiting on slow deployments, output drops. Stripe's *Developer Coefficient* study found developers lose more than 17 hours a week to maintenance and bad code, with roughly $300 billion in annual GDP impact globally.[7] The cumulative effect is not just slower delivery, but a frustrating day-to-day experience.

Solving this needs investment in toolchains that fit together: standardized core environments, modern CI/CD practices, and continuous tuning.

Cognitive load and developer burnout

Enterprise software development is inherently complex. Engineers juggle multiple priorities, tools, and stakeholders. When documentation is sprawling, knowledge sits in scattered repositories, and information is locked in silos, developers spend hours searching for answers or working from outdated docs instead of writing code.

Frequent context switching compounds the problem. Multiple projects, urgent requests, and back-to-back meetings leave little room for focused work. Constant interruption breaks concentration, slows progress, and lowers code quality.

Pressure to ship quickly without adequate support pushes engineers toward burnout. Aggressive deadlines combined with inefficient processes and weak tooling drain engagement and degrade performance. Atlassian and DX's 2024 State of Developer Experience survey of more than 2,100 developers and engineering leaders found that 69% of developers lose at least eight hours per week to inefficiencies, with technical debt and inadequate documentation as the leading causes.[8] Research by Graziotin et al. confirms what experienced engineering leaders see in practice: developer happiness directly affects code quality, productivity, and team dynamics.[9] The pattern compounds over time: more frustration, less output, higher attrition.

Reducing cognitive load takes a cultural shift toward sustainable work. That means easy-to-navigate documentation, fewer unnecessary context switches, and realistic timelines.

Tackling these three challenges, organizational inefficiencies, fragmented tooling, and cognitive overload, gives enterprises the foundation for a more effective and rewarding developer experience.

The next section moves from problems to action: practical strategies for transforming DevEx and giving engineers the conditions to do their best work.

A framework for improving DevEx

This section sets out a framework with three pillars to enhance DevEx:

- Streamlining workflows and reducing bureaucracy
- Investing in tooling and infrastructure
- Building a supportive engineering culture

Streamlining workflows and reducing bureaucracy

Improving DevEx starts with attacking the inefficiencies built into workflows and organizational processes. Bureaucracy and administrative overhead drag down development and disengage the engineers doing the work. Streamlining workflows and removing unnecessary barriers gives developers room to focus on what they're hired to do: write high-quality code.

Adopting agile and DevOps practices is central to this. Agile keeps work flexible, collaborative, and tied to actual feedback, so development stays aligned with business goals. DevOps brings automation and continuous delivery, cutting handoff delays and operational drag. The DORA research program, summarized in Forsgren, Humble, and Kim's Accelerate, found that teams investing in these practices ship more often, recover faster, and report less burnout.[1] Together they reduce the friction baked into the software development lifecycle.

Cutting administrative overhead is the next lever. Developers routinely lose time to compliance forms, low-value meetings, and approval queues. Automating compliance checks, tightening approval workflows, and dropping meetings that don't earn their place returns hours to engineering work.

Decision-making autonomy makes the rest stick. When developers have real authority over technical decisions, they move faster, experiment more, and own the outcomes. The job of leadership is to set up the conditions for that, building trust and accountability instead of layering on oversight.

Investing in tooling and infrastructure

Once workflows are cleaner, the tooling has to keep up. Outdated or poorly connected tools throttle output and frustrate engineers. McKinsey's Developer Velocity research, which surveyed senior executives at 440 enterprises, found best-in-class tools to be the single largest driver of software-related business performance, ahead of culture, product management, and talent.[10]

Strong DevEx depends on a tool stack where things connect cleanly, cognitive load stays low, and friction gets engineered out.

The first piece is integration. Development environments, CI/CD pipelines, code repositories, and collaboration tools should connect cleanly so developers aren't troubleshooting compatibility issues or hopping between disconnected systems. Modern platforms with API-driven integrations and plug-and-play functionality are worth the investment.

Self-service is where the time savings get real. When developers can spin up environments, deploy applications, and pull resources without filing a ticket, the calendar opens up. Internal developer portals that surface APIs, cloud resources, and documentation in one place mean engineers stop waiting on other teams.

Automation handles what's left. Code formatting, testing, deployment, all of it can move to scripts and pipelines, freeing developers for work that actually needs a human. Infrastructure as Code (IaC), automated testing frameworks, and AI-assisted coding tools all play a role in reducing manual effort and inconsistency.

Building a supportive engineering culture

Tools and processes only go so far. Culture shapes how DevEx actually feels day to day, and it's where retention, engagement, and innovation are won or lost. Enterprises need to build environments where developers feel valued, heard, and supported.

Psychological safety is the foundation. Developers should be able to raise ideas, try new approaches, and admit mistakes without worrying about consequences. Open feedback, blameless post-mortems, and leadership that listens are how that culture takes root.

Collaboration and knowledge sharing are equally important. Engineering teams do their best work when information flows freely, and people learn from each other. Regular technical discussions, strong documentation practices, and internal mentorship turn individual expertise into shared capability.

Balancing performance expectations with well-being keeps the system sustainable. High expectations drive innovation, but constant pressure pushes people toward burnout. Monitoring workload, building in flexibility, and resourcing real support, mental health, growth paths, and wellness are what separate a team that lasts from one that churns.

What supports a better DevEx? Table 1 highlights a number of engineering practices to support and improve DevEx in the enterprise.

TABLE 1

Engineering practices to support and improve DevEx

PRACTICE	DESCRIPTION
Invest in CI/CD	Automate code integration, testing, and deployment pipelines to streamline development, reduce manual errors, and ship faster, more reliable releases.
Create internal developer portals (Confluence pages)	Provide a centralized knowledge hub with documentation, best practices, and tools to improve onboarding, collaboration, and productivity.
Modularize the codebase	Break the codebase into smaller, reusable modules to make development, testing, and maintenance easier, enabling faster iteration.
Activate developer feedback loops (code reviews and regular retrospectives)	Establish continuous feedback through code reviews and retrospectives to improve code quality, share knowledge, and build a learning culture.

PRACTICE	DESCRIPTION
Deliver training and tech talks	Offer continuous learning through training sessions and internal tech talks to keep developers current on new technologies and foster knowledge sharing.
Establish clear onboarding processes	Provide a well-structured onboarding process so new developers ramp up quickly and have access to the right resources and support.
Adopt code quality tools and standards	Implement linters, formatters, static analysis tools, and shared coding standards to maintain quality and reduce manual rework.
Provide self-service infrastructure	Allow developers to manage their environments and tools (cloud resources, database setups) through self-service portals, reducing friction and increasing output.
Improve developer tools and IDEs	Ensure developers have access to high-quality IDEs, debugging tools, and integrations that streamline coding, testing, and deployment.
Automate testing (unit, integration, E2E)	Invest in automated testing at every level to catch bugs early, raise code quality, and reduce manual testing effort.
Easy access to production metrics and logs (dashboards)	Give developers direct access to real-time production metrics and logs so they can spot issues and optimize performance quickly.
Low-cost experimentation and prototyping	Enable rapid experimentation with new features or ideas through a flexible prototyping environment with minimal overhead.
Internal knowledge sharing platforms	Create spaces (wikis, forums) where developers share knowledge, solutions, and experiences across teams.
Focus on developer well-being	Promote work-life balance and mental health support (flexible hours, wellness programs, stress management resources).
Adopt practical collaboration tools (Slack, Jira, etc.)	Use collaboration tools that help teams communicate, track issues, and coordinate across projects, reducing silos.
Enable cross-functional team collaboration	Foster collaboration between development, operations, product, and other teams to align on goals, sharpen feedback cycles, and accelerate delivery.
Establish and communicate clear and transparent roadmaps	Share roadmaps with developers so they understand priorities and can plan their work, improving alignment and focus.
Implement automated code deployment with rollbacks	Build automated deployment pipelines with rollback capabilities so issues in production can be fixed quickly and safely.
Use feature toggles for incremental releases	Use feature toggles to enable incremental releases and tests, rolling out features progressively without disrupting the system.
Offer comprehensive API documentation	Maintain clear, current API documentation so developers can integrate with internal services without confusion or delay.
Provide observability and monitoring tools	Equip developers with observability and monitoring tools (Prometheus, Grafana) to track system health, catch issues early, and tune performance.
Establish code ownership and responsibility	Set up code ownership where developers own specific parts of the codebase, building accountability and improving quality.
Define transparent performance metrics and KPIs	Share relevant performance metrics and KPIs with developers to encourage a results-driven approach and let them tune their workflows.
Offer cloud-based development environments	Provide cloud-based development environments that are easy to set up and scale, so developers can work effectively from anywhere.
Foster a culture of experimentation	Encourage experimentation and risk-taking, where developers can test hypotheses, try new ideas, and learn from what doesn't work.
Integrate with third-party tools and services	Ensure the development environment connects well with third-party tools, services, and APIs to streamline workflows.
Establish clear deployment and release processes	Establish transparent, standardized deployment and release processes so developers can ship without surprises.
Customize developer environments	Allow developers to customize their environments (IDE settings, terminal configurations) to match their workflows, increasing comfort and output.
Enable mentorship and pair programming	Encourage mentorship and pair programming to transfer knowledge, build skills, and strengthen team relationships.
Encourage autonomy and ownership	Empower developers to own their projects and decisions, building responsibility and motivation to deliver quality work.

These practices (Table 1), such as improved CI/CD, internal knowledge hubs, modular codebases, and continuous feedback, build a more efficient, collaborative, and sustainable development environment. None of it will be sustained unless evaluation systems and incentives are explicitly designed to support and protect them.

Leadership incentives and the measurement lag

The three pillars only hold if leadership is measured on building the environment, not only on the output that environment eventually produces. That distinction tends to break at the middle-management layer. A director judged on this quarter's delivery has every reason to defer tooling work, treat the experimentation budget as slack, and let DevEx wait for a quarter. The fix is to make whoever owns DevEx accountable for building and holding the environment first, and to judge them on outcome metrics only once that environment has had time to take. There is therefore a real risk that an effective improvement initiative will be terminated just before it begins to produce returns.

The payoff lags the investment. Investments in workflow, tooling, and culture, the three pillars above, register first as lower developer-perceived friction and only later as stronger delivery numbers, which is why the DevEx research treats perception measures as leading indicators of performance.[3, 4] Judged too early, the outcome KPIs make good work look like failure just before the returns arrive. That makes measurement the next question: which metrics to track, and how to read them without punishing the work that takes time to pay off.

Key metrics and KPIs for measuring DevEx

Enterprises need clear metrics to measure DevEx and keep improving it. Without quantifiable insight, decisions get made on assumptions rather than evidence, and improvements miss what developers actually need.

The metrics that matter fall into six areas: engineering flow, developer satisfaction, collaboration, code quality, productivity, and infrastructure and tooling. Each gives a different view of how DevEx is performing, and together they provide the full picture. This grouping draws on three established frameworks: DORA's four key metrics for delivery performance,[1, 11] the SPACE framework for multidimensional productivity measurement,[2] and the DevEx framework for capturing developer-perceived friction.[3, 4]

It's important to state two recommendations before we get to the tables.

First, we recommend you read these as trendlines, not pass-or-fail thresholds. The important question is whether a team is getting faster and more reliable over time, not whether it clears a fixed bar this quarter.

Secondly, we recommend you treat the numbers as signals for teams, not scorecards for individuals.

The SPACE framework argues that productivity is multidimensional and cannot be reduced to a single metric, and it cautions against using these measures to rank individuals.[2] DORA frames delivery performance the same way, as movement over time across performance cohorts rather than absolute targets.[1, 11] It is well understood that experimentation needs explicit room. When teams try new approaches, speed and quality often dip before they recover, so holding experimental projects and process changes to the same flow and quality expectations discourages the risk-taking this report argues for. Therefore, it is critical to give that work leeway and judge it over a few cycles on what it produces and what it teaches, not on its first-quarter numbers.

Engineering flow metrics

Engineering flow metrics measure the speed and reliability of the development pipeline, showing how quickly software moves from idea to production. The first four metrics in Table 2 are the DORA "four key metrics," validated through more than a decade of State of DevOps research as the strongest predictors of software delivery performance.[1, 11]

Table 2 summarizes the engineering flow metrics.

TABLE 2

Engineering flow metrics

METRIC	DESCRIPTION
Lead time for changes	Time from code commit to deployment, reflecting the speed of the development pipeline.
Deployment frequency	How often code is deployed to production, indicating the efficiency of release cycles.
Mean time to recovery (MTTR)	Average time to restore the system after a failure, representing resilience and recovery capability.
Change failure rate	Percentage of changes that result in failures, indicating release stability.
Cycle time	Time from when work starts on a feature to when it is finished, showing development efficiency.
Code churn	Frequency of code changes after initial commit, which can indicate rework or instability.
Pull request review time	Average time for code reviews and merging pull requests, affects team collaboration speed.
Build time	Average time for builds to complete, affecting feedback loops and productivity.

Developer satisfaction and feedback metrics

These metrics gauge developer morale, satisfaction, and the effectiveness of the processes that support growth and retention. The SPACE framework places satisfaction and well-being at the same level as performance and activity, arguing that productivity cannot be captured without measuring how developers feel about their work.[2] The DevEx framework reinforces this, showing that developer perceptions of friction and flow are leading indicators of delivery performance.[3, 4]

TABLE 3

Developer satisfaction metrics

METRIC	DESCRIPTION
Employee Net Promoter Score (eNPS)	How likely developers are to recommend their workplace to others, reflecting overall satisfaction.
Survey-based satisfaction scores	Regular developer surveys covering work satisfaction, tool effectiveness, and team morale.
Attrition rate	The percentage of developers leaving the organization is an indicator of dissatisfaction or unmet needs.
Time to onboard	Time for a new developer to become productive, measuring onboarding effectiveness.
Training and skill development rate	The time developers spend on training and acquiring new skills reflects investment in growth.
Burnout rate	The frequency of developer burnout often correlates with workload and a lack of work-life balance.
Retention rate	The percentage of developers staying in the organization indicates job satisfaction and growth opportunities.

Collaboration and communication metrics

Collaboration and communication metrics (Table 4) track how effectively teams collaborate, communicate, and share knowledge, all central to DevEx.

TABLE 4

Collaboration and communication metrics

METRIC	DESCRIPTION
Cross-department collaboration rate	How frequently and effectively developers collaborate with non-technical teams (product, marketing).
Communication latency	The time it takes for communication to occur between teams and within the development team.
Number of pair programming sessions	The frequency of pair programming, which builds collaboration and knowledge sharing.
Team alignment	The extent to which teams are aligned with project goals through regular feedback and reviews.
Internal documentation quality	How well internal resources (code, wikis) are maintained affects collaboration and knowledge sharing.
Meeting effectiveness	Number of meetings and time spent, helping evaluate communication efficiency.
Knowledge sharing rate	How frequently and effectively knowledge is shared across teams (documentation, tech talks).
Peer review participation rate	Level of participation in peer reviews, reflecting collaboration and code quality.

Code quality and technical metrics

Code quality metrics (Table 5) assess the quality, maintainability, and stability of the codebase, showing how reliably the development process produces dependable software.

TABLE 5

Code quality metrics

METRIC	DESCRIPTION
Defect density	Number of bugs or issues per line of code or feature, indicating code quality.
Test coverage	Percentage of the codebase covered by automated tests, supporting performance, reliability, stability, and robustness
Code review acceptance rate	Percentage of code passing review on the first submission, indicating quality and reviewer effectiveness.
Static code analysis results	Code quality assessment from automated tools, surfacing security vulnerabilities, and standards violations.
Technical debt	Accumulated cost of maintaining suboptimal code, reflecting the need for refactoring and long-term code health.
Bug resolution time	Time to resolve bugs, identifying where development processes need improvement.

Developer productivity metrics

Developer productivity metrics (Table 6) measure how efficiently developers produce quality code, balancing output with quality and team alignment.

Infrastructure and tooling metrics

Infrastructure and tooling metrics (Table 7) track the reliability and effectiveness of the tools and infrastructure developers depend on, with direct impact on workflow and output.

These metrics give a full view of DevEx across multiple dimensions, but no organization needs to track all of them. The recommendation is to pick KPIs from each of the identified key DevEx pillars that map to your priorities, goals, and where the biggest friction lives, and execute on them.

TABLE 6

Developer productivity metrics

METRIC	DESCRIPTION
Number of features delivered	Features delivered over a specific period, indicating team output and focus.
Lines of code (LOC)	The amount of code written is useful as a directional signal, but only when paired with quality measures.
Task completion rate	Percentage of tasks completed in a sprint or timeframe, reflecting team efficiency.
Code reusability rate	Extent to which code is reused across projects, indicating efficiency and maintainability.
Time spent on non-coding activities	The time developers spend outside of direct coding (meetings, documentation) affects overall output.
Feature adoption rate	How often new features are used in production, indicating whether features meet user needs.

TABLE 7

Infrastructure and tooling metrics

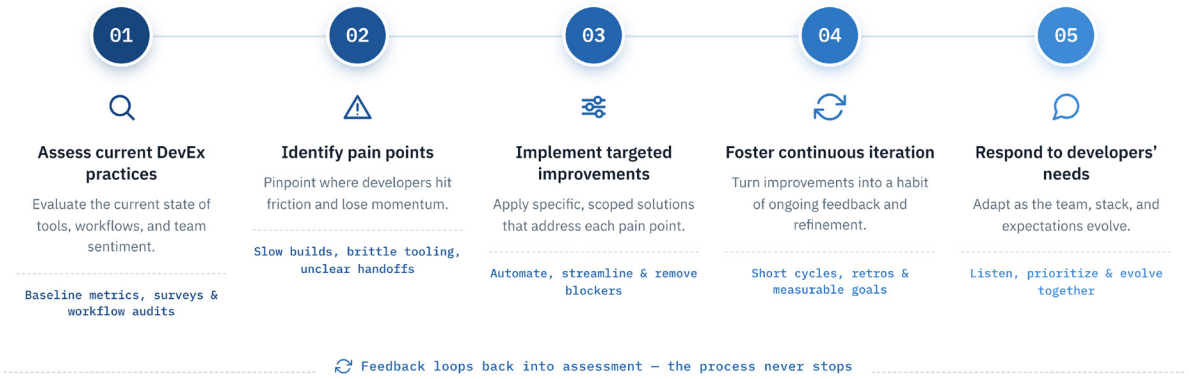
METRIC	DESCRIPTION
Tool integration time	Time to integrate new tools or services into the development pipeline, affecting DevEx.
Availability and uptime of dev tools	Reliability and uptime of key developer tools (CI/CD systems, IDEs, version control) affect workflow efficiency.
Dev environment provisioning time	Time to provision a new development environment, affecting setup time and developer efficiency.
Tool satisfaction rate	Developer surveys assessing satisfaction with the tools they use, identifying areas for improvement.
Self-service portal usage	Usage rate of internal self-service tools for provisioning environments, managing dependencies, and accessing resources.

Conclusion

Improving DevEx had a direct impact and pays dividends. Organizations that invest in it and follow the DevEx improvement process (Figure 1) ship faster, build better products, keep their senior talent longer, and adjust to market shifts more quickly[4, 11].

FIGURE 1

The DevEx improvement process is a continuous five steps approach to reducing developer friction



Product quality follows from the same investments. When developers have the right tools, clean workflows, and a culture that backs them, they produce code with fewer defects, less technical debt, and less costly rework. Organizations that take DevEx seriously also become more attractive places to work, which lowers recruitment and onboarding costs and keeps engineering teams stable.

The actions are concrete: assess current DevEx, identify where the friction is worst, and implement targeted improvements. Over time, building a culture of continuous iteration around developer needs is what makes the gains sustainable.

Enterprises that lead in DevEx will move faster, ship better products, and hold onto stronger engineering teams. The ones that don't will find their best people leaving for competitors who took it seriously.

References

1. Forsgren, N., Humble, J., Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps: Building and Scaling High-Performing Technology Organizations. IT Revolution Press.

2. Forsgren, N., Storey, M.A., Maddila, C., Zimmermann, T., Houck, B., Butler, J. (2021). "The SPACE of Developer Productivity: There's more to it than you think." ACM Queue, Vol. 19, No. 1, pp. 20-48.

3. Noda, A., Storey, M.A., Forsgren, N., Greiler, M. (2023). "DevEx: What Actually Drives Productivity." ACM Queue, Vol. 21, No. 2, pp. 35-53.

4. Forsgren, N., Kalliamvakou, E., Noda, A., Greiler, M., Houck, B., Storey, M.A. (2024). "DevEx in Action: A study of its tangible impacts." ACM Queue, Vol. 21, No. 6.

5. Greiler, M., Storey, M.A., Noda, A. (2022). "An Actionable Framework for Understanding and Improving Developer Experience." IEEE Transactions on Software Engineering, 49(4), 1411-1425.

6. Skelton, M., Pais, M. (2019, 2nd ed. 2025). Team Topologies: Organizing Business and Technology Teams for Fast Flow. IT Revolution Press.

7. Stripe & Harris Poll (2018). The Developer Coefficient: Software engineering efficiency and its $3 trillion impact on global GDP.

8. Atlassian & DX (2024). State of Developer Experience Report 2024. In partnership with Wakefield Research.

9. Graziotin, D., Fagerholm, F., Wang, X., Abrahamsson, P. (2018). "What happens when software developers are (un)happy." Journal of Systems and Software, Vol. 140, pp. 32-47.

10. Srivastava, S., Trehan, K., Wagle, D., Wang, J. (2020). Developer Velocity:z How software excellence fuels business performance. McKinsey & Company.

11. DORA / Google Cloud (2024). Accelerate State of DevOps Report 2024.

Feedback

This report has been prepared to the best of the author's ability. Despite careful review, typographical errors or inaccuracies may remain. If you spot an error or have a suggested correction or improvement, please contact the **author** directly.

Disclaimer

The views expressed in this report are those of the author alone. They do not necessarily reflect the views of any organization or employer with which the author is currently or has previously been affiliated.

Acknowledgments

The author thanks Linux Foundation Research, **Hilary Carter** and team for their review, feedback, and producing the report and infographics. Particular thanks go to **Arun Gupta** for the Foreword and the running metaphor that had captured the essence of the report, **Shu Muto** for the detailed feedback and suggestions especially on using more descriptive language, and to **Wesley Faulkner**, whose detailed review sharpened the report's treatment of developer experience metrics, experimentation, and the incentive structures that sustain a strong engineering environment.

The author also recognizes the Linux Foundation and the DEVREL Foundation for publishing this report and for their continued leadership in advancing the field.

About the author

Ibrahim Haddad, Ph.D., is Head of Infotainment Engineering at Volvo Cars, leading engineering for the company's next-generation in-vehicle software platform. Prior to Volvo Cars, he served as Vice President of AI Strategic Programs at the Linux Foundation, leading LF AI & Data and growing it into the largest neutral open source AI ecosystem, and concurrently as Founding Executive Director of the PyTorch Foundation.

At Samsung, he served as Distinguished Engineer and VP of R&D.

Haddad's career spans executive and technical leadership roles at Volvo Cars, Samsung, the Linux Foundation, Ericsson, Motorola, Palm, and Hewlett-Packard. He has championed open source driven by the belief that collaborative development is a faster, better, and cheaper path to innovation. He is a recognized expert in open source license compliance and a long-standing advocate for OSPOs as the structural backbone for external R&D engagement.

Haddad has authored 7 books, 21 e-books, and more than 150 publications covering open source strategy, AI governance, engineering leadership, and ecosystem development. He has advised the World Bank Group and presented to the European and UK Parliaments on open source and AI. He earned his Ph.D. with honors in Computer Science from Concordia University, where he was awarded both the J. W. McConnell Memorial Graduate Fellowship and the Concordia University 25th Anniversary Fellowship.

LinkedIn | Portfolio

x.com/linuxfoundation

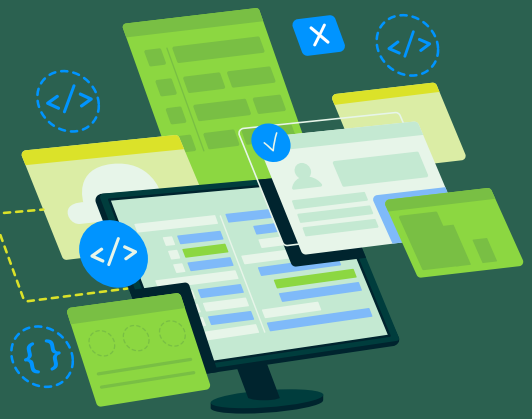
facebook.com/TheLinuxFoundation

linkedin.com/company/the-linux-foundation

youtube.com/user/TheLinuxFoundation

github.com/linuxfoundation

September 2026



To reference this work, please cite as follows: Ibrahim Haddad, "Optimizing Developer Experience for Enterprise Outcomes: A Roadmap to Reducing Friction, Measuring Impact, and Scaling Productivity in Software Delivery," Foreword by Arun Gupta, Linux Foundation, September 2026.

THE LINUX FOUNDATION | Research

Founded in 2021, Linux Foundation Research explores the growing scale of open source collaboration, providing insight into emerging technology trends, best practices, and the global impact of open source projects. Through leveraging project databases and networks, and a commitment to best practices in quantitative and qualitative methodologies, Linux Foundation Research is creating the go-to library for open source insights for the benefit of organizations the world over.