Mastering Modern DevOps Performance

An Intellyx Analyst Guide for Faros AI

By Jason Bloomberg and Jason English, Intellyx

©2024 Intellyx B.V.

Table of Contents

Introduction

The concept of measuring the performance of software development teams is nothing new. Productivity has never been more important, as the companies that fail to develop new features to meet customer needs quicker than their competition, will likely quickly fail.

Unfortunately, different teams and individual engineers have entirely different ideas of what performance means. How can we understand the speed and quality tradeoffs of balancing efficiency and innovation, while eliminating unproductive, morale-killing toil from software development orgs? Perhaps AI-augmented DevOps processes can get us into a better rhythm and flow.

This 4-part Intellyx Analyst Guide will help readers understand how to master DevOps performance objectives.

By Jason English

Director & Principal Analyst
Intellyx

Does measuring software engineering performance actually deliver value?

Part 1 of the Mastering Modern DevOps Performance Series

Every enterprise in the world wants to maximize performance: delivering for customers better, faster, and cheaper than the competition.

Further, software company executives love to repeat the mantra that "every company is a software company" as often as possible.

Therefore, it stands to reason that management consulting firms would seek to apply their MBA statistical models to maximize performance of the software-producing function of any enterprise.

The concept of measuring the performance of software development teams is nothing new, but it recently returned to the public consciousness with a little controversy thanks to this recent McKinsey piece titled: *"Yes, you can measure software developer productivity."*

Implement their methodology, the article says, and developers could realize a 20-to-30 percent reduction in customer-reported defects, a 20 percent improvement in employee experience scores, and a 60 percent improvement in customer satisfaction.

Sounds incredible! With results like that, why hasn't everyone already jumped on their proposed measurement bandwagon?

Why measure developer productivity?

Compared to other process-oriented industries, the software industry has been rather undisciplined in its approach to measuring results. An ineffable 'tiger team' mentality arose, where we expected one genius developer or an expert team to lock themselves in the office with a couple pizzas and some Jolt Cola, and hammer out brilliant code.

This 'code cowboy' mentality predictably led to failure and heartbreak, as two-thirds of software projects consistently failed to meet budgets and timelines.

CEOs and CFOs were constantly frustrated by a lack of accountability. They wanted engineering orgs to take a page from the discipline of industrial supply chain optimization, so software development could realize the benefits of KPI measurements, Kanban-style workflows, and process automation that built everything else in our modern economy.

The DevOps movement evolved from Agile methodologies around 2008, and engineering organizations started looking at software delivery through a continuous improvement lens. We learned to empower dev teams to collaborate with empathy while 'measuring what matters' and 'automating everything' toward delivering customer value.

The release of *The Phoenix Project* book articulated the connection between DevOps and supply chain optimization, highlighting the Three Ways: flow/systems thinking, feedback loops, and a culture of continuous improvement reminiscent of the best-running Toyota car factories in Japan.

In an industrial supply chain scenario, planners could look for signals like supplier availability, work-in-process, and inventory turns as performance indicators. By comparison, software development deals with much less substantial signals — bits and bytes moving over the internet: the intellectual assets of ideas, requirements, and data.

If we are to achieve a new wave of industrialization in the software industry, clearly coming to grips with the data that feeds the software supply chain is our first priority.

Where measurements meet incentives

The McKinsey model was built atop two currently popular frameworks: DORA (DevOps Research and Assessment) metrics, popularized by Google and many other companies invested in the DevOps movement; and SPACE metrics (satisfaction, performance, activity, communication and collaboration, and efficiency) added by GitHub and Microsoft.

On top of that, they added a set of new 'opportunity focused' metrics: Developer velocity benchmarks, contribution analysis, talent capability score, and inner/outer loop time spent.

Interestingly, their "inner/outer loop" metric uniquely prioritizes time spent on the "inner loop" building (coding and testing) software, instead of the "outer loop" time spent on integration, integration testing, releasing, and deployment.

But what if that outer loop is a vitally important part of certain roles in the engineering org? To avoid technical debt, we need architects focused on system design, and SREs capable of tracking down root causes of issues in deployment.

This wonderfully vitriolic blog response in The Pragmatic Engineer with Kent Beck and Gergely Orosz responds with a perfect example of how a measurement initiative that started with decent results eventually strayed:

"At Facebook we [Kent here] instituted the sorts of surveys McKinsey recommends. That was good for about a year. The surveys provided valuable feedback about the current state of developer sentiment.

Then folks decided that they wanted to make the survey results more legible so they could track trends over time. They computed an overall score from the survey. Very reasonable thing to do. That was good for another year. A 4.5 became a 4. What happened?

Then those scores started cropping up in performance reviews, just as a "and they are doing such a good job that their score is 4.5". That was good for another year.

Then those scores started getting rolled up. A manager's score was the average of their reports' scores. A director's score would be the average of their reporting managers' scores.

Now things started getting unhinged. Directors put pressure on managers for better scores. Managers started negotiating with individual contributors for better survey scores. "Give me a 5 & I'll make sure you get an 'exceeds expectations'." Directors started cutting managers & teams with poor scores, whether those cuts made organizational sense or not."

Whoa. How orgs act upon development metrics is as important as the measurements themselves. Nobody wants to see performance improvement goals create a zero-sum game that disheartens valued technical talent.

On the positive side, McKinsey's article can only spur more thought and discussion among the development community toward how engineering orgs can deliver more predictable metrics, like the ones CEOs and CFOs expect to see from other groups like sales and customer services.

Developer enablement metrics for success at Autodesk

You already know Autodesk—if you've ever seen a really cool modern building, or a hyper-realistic 3D animated film, chances are, their software was used by professionals to help design or create it.

Autodesk supports a suite of highly refined and specialized CAD and design tools, but as they started migrating to a common cloud-and-microservices-based architecture to improve scalability and automate deployment infrastructure, delivery time became unpredictable, with teams stymied by environment availability and service interdependencies.

"If ten teams are doing well and only one team is doing poorly, you are only as good as your weakest link," said Ben Cochran, VP of the newly formed Developer Enablement team, reporting directly to the CTO.

With an eye to improving developer experience and morale across their system, rather than at an individual level, the team adopted DORA metrics, including deployment frequency, mean time to recovery (MTTR), lead time, and change failure rate (CFR) as Autodesk's foundation for productivity measurement.

The output velocity and business outcomes of their software team were improved, but in the macro view, creating an environment of collaboration and shared learning that removes roadblocks, rather than taking punitive measures based on measurements, made all the difference.

The Intellyx Take

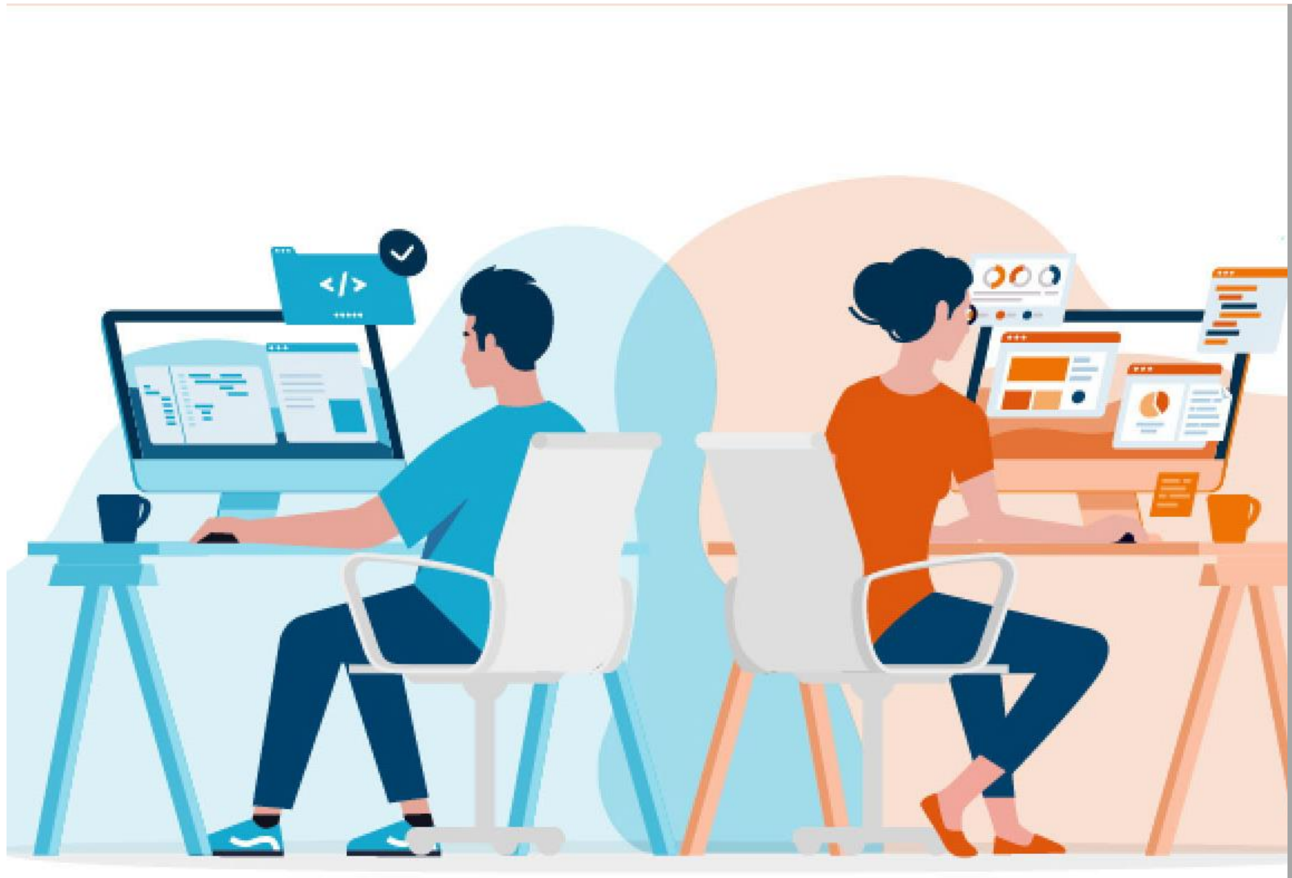
For engineers, too much emphasis on monitoring and metrics can feel like Big Brother is looking over your shoulder, inhibiting creative problem solving. Conversely, a lack of measurement also means that problems aren't getting reliably solved.

Poor development performance metrics overlook the constant competitive imperative for achieving more productivity with fewer resources, and can eventually result in layoffs or draconian performance measures being put in place.

Success at measurement depends on a balancing act between innovation and efficiency, while aligning team members with high-value business outcomes and eliminating administrative toil from the development process.

Even if there's healthy disagreement about the details of McKinsey's developer performance model, it's useful to get everyone talking about how to mature the discipline of software development.

Said Vitaly Gordon, CEO of Faros.ai in a recent blog: *"McKinsey speaks the language of the C-Suite well. If they can get executives to commit time and effort to removing friction from the engineering experience based on what the data is telling us, I am all for it."*



By Jason Bloomberg

Managing Director & Analyst
Intellyx

Avoiding the Developer Productivity Paradox

Part 2 of the Mastering Modern DevOps Performance Series

Avoiding the Developer Productivity Paradox

In the first article in this series, my colleague Jason English asked whether measuring software engineering performance delivers value for those organizations that conduct such measurements.

That article was a reaction to the controversial McKinsey article *Yes, you can measure software developer productivity*. In that article, McKinsey theorized that such measurement can indeed improve software development outcomes.

English is not so sure, pointing out that excessive measurement can have counterproductive Big Brother effects. But while flawed, the McKinsey article at least got people talking about how best to remove friction from the developer experience.

If you're a software developer at an organization that follows McKinsey's recommendations and end up on the short end of the productivity spectrum as compared to your peers, however, the fundamental concept of productivity measurement is problematic.

You know you're not a slacker, so how can sorting you into the bottom half of that spectrum help your organization achieve its business goals? Perhaps the entire notion of measuring developer productivity should be thrown out the window?

Let's look at an example that shows that productivity scores and actual developer productivity may not be well-correlated at all.

When Less is More

Let’s say an organization has two developers on its team. Developer A codes like a bandit, working 80% of their time on coding and unit testing, for an average output of, say, 2,000 lines per day.

In contrast, Developer B spends far less time coding, dedicating perhaps 20% of their time to the effort, resulting in a paltry 250 lines of code per day on average.

Which developer is more productive?

At first glance, it looks like Developer B is slacking off. Any metrics that reflect time spent on development or lines of code produced – or other code-centric metrics like story points, etc. – would clearly rank Developer B lower than Developer A.

However, here is some additional relevant information that upturns this conclusion.

- Developer B is far more senior than Developer A. Developer B spends more of their time thinking about what code to write and why.
- Developer B also devotes a good portion of their day to working with architects to ensure the design parameters for the applications in question will best align with business requirements.
- Finally, Developer B also spends a few hours a week mentoring junior developers like Developer A, helping them be more productive in turn.

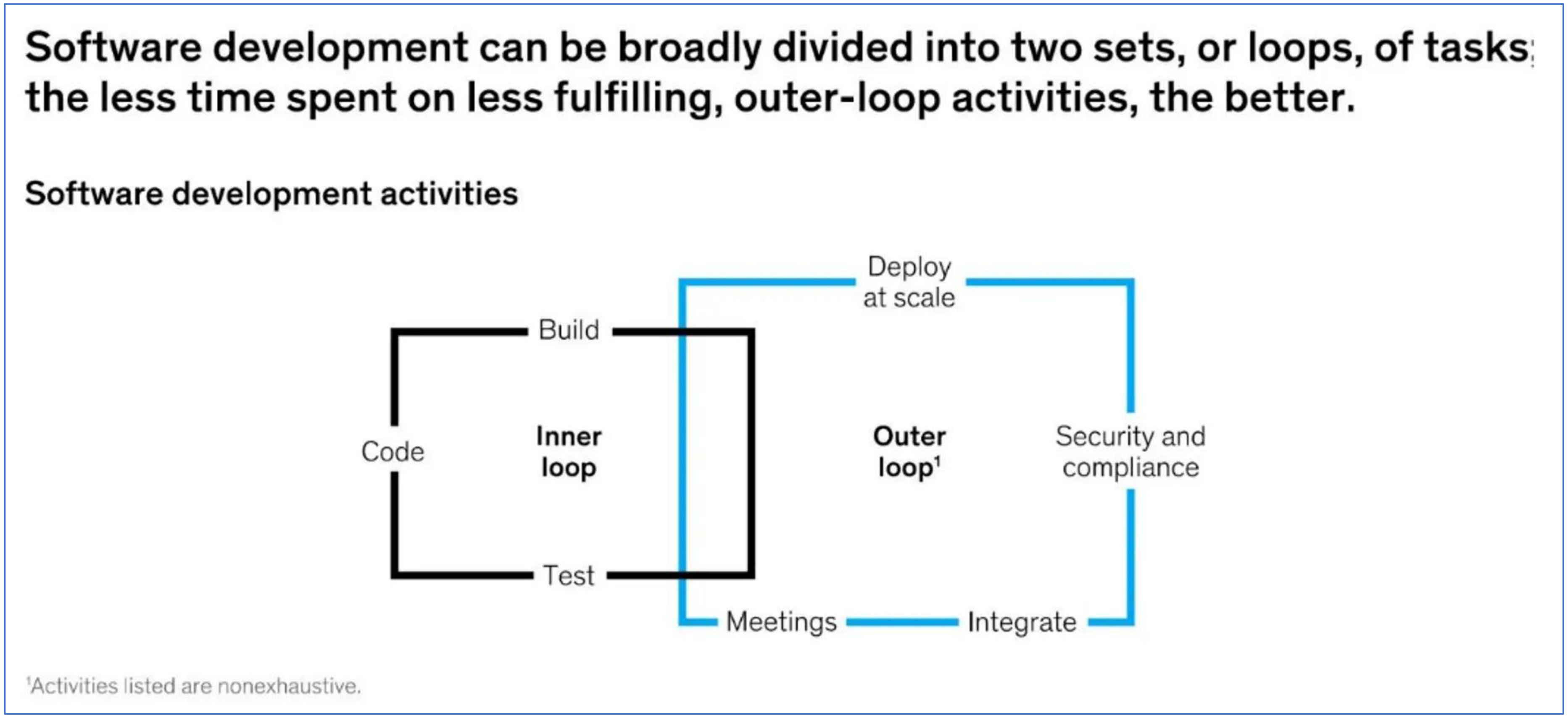
Developer A, in contrast, is doing their best to generate quantity over quality to show how productive they are.

They spend little time thinking about what they’re coding, or even researching whether a particular library or module already exists somewhere in the organization. As a result, they generate a lot of redundant or otherwise useless code.

Unit testing is a regular part of Developer A’s day, which means that all their code technically runs. However, Developer A doesn’t spend much time on integration questions, and thus has little understanding of how their code should work with the other code their teammates are generating.

McKinsey Misses the Big Picture

McKinsey's analysis of developer productivity breaks down software development into two sets of tasks, as the diagram below from the article in question illustrates.



Software development can be broadly divided into two sets, or loops, of tasks; the less time spent on less fulfilling, outer-loop activities, the better.

Software development activities

[1]Activities listed are nonexhaustive.

McKinsey's two sets of development tasks (Source: McKinsey)

According to McKinsey, the inner loop above – build, code, test – should be how developers ideally spend their time. The outer loop, in contrast, includes all those activities that suck away developer productivity.

Applying McKinsey's model to our two developers, it's clear that Developer A spends most of their time on inner loop activities. Good for them!

Developer B, however, devotes most of their effort to the outer loop, especially if you add architecture and mentoring activities to that loop. (McKinsey's footnote points out that tasks are missing from the diagram. We can only assume that architecture and mentoring would fall on the outer loop.)

Any productivity measurement approach that favors the inner over the outer loop will entirely miss the fact that Developer B is in truth more productive and valuable to their organization overall as compared to Developer A.

Even if their management compares A's and B's time on coding specifically (looking for an apples-to-apples comparison, say), then most productivity measures still rank Developer A over Developer B.

Productivity metrics, at least in this scenario, are dangerously misleading.

The Big Picture of Developer Productivity

The key takeaway here is that blindly focusing on individual productivity metrics without considering the roles and responsibilities of developers with different levels of seniority doesn't accurately reflect the productivity of the team – or the development organization at large.

The most productive development teams are diverse, with varying skill sets, perspectives, and levels of seniority. Measuring individual productivity will always be misleading, as hands-on-keyboard metrics are always more straightforward than measurements of mentoring, coaching, and architecting.

Software engineering intelligence platforms like Faros AI can help engineering managers and their bosses get a handle on team and group productivity, including these difficult-to-measure tasks that are so critical for software development success.

The Intellyx Take

This article has only scratched the surface of the issues inherent in measuring developer productivity.

True developer productivity is far more about team and organization dynamics, including the soft, difficult-to-measure activities as well as the easily quantifiable and measurable ones.

I'm not saying that measuring developer productivity is pointless. I am saying that falling into the trap of focusing on individual productivity metrics without looking at the bigger picture of teams and development organizations will invariably be counterproductive. Don't make that mistake.

By Jason Bloomberg

Managing Director & Analyst
Intellyx

Optimizing the Software Velocity vs. Safety Tradeoff

Part 3 of the Mastering Modern DevOps Performance Series

"If everything seems under control, you're not going fast enough."

— Mario Andretti

When we drive our cars, safety is paramount. We moderate our speed, use our mirrors, and drive defensively. If conditions require us to slow down, then we slow down.

Unlike legendary racecar driver Mario Andretti, our goal is to get to our destination as safely as possible.

For Andretti, however, the goal is to win the race. Safety is but a means to an end, as crashing is a surefire way to lose.

This contrast between the two extremes of automobile driving has a close analog in software development.

For software, safety refers to reliable, bug-free code. Sometimes safety is paramount, like with bank transaction processing or satellite software. In such situations, delivering code that is optimally reliable is the main goal, and if it takes more time to deliver it, then so be it.

In other situations, software velocity is a top priority, for example, with web-based companies or digital offerings in general. These organizations' competitiveness – and thus, their survival – depends upon delivering changes to code quickly.

Both perspectives are valid, as they both focus on managing the risks inherent in software development – the risks of delivering broken code vs. the risk of delivering code too slowly to meet the competitive requirements of the business.

What, then, is the best way of trading off velocity and safety? Once we answer that question, then another question becomes paramount: how can we improve both velocity and safety at once? Understanding the tradeoff is one thing, but we really want both at the same time.

After all, that's how Mario Andretti won his races – and lived to race another day.

Shifting the Velocity/Safety Balance Point

The only way we'll avoid the pitfalls inherent in trading off velocity and safety is to manage software development risk across the board.

At some point, the development organization reaches the optimal tradeoff. Conventional wisdom says that this tradeoff is the best that development teams can achieve. After all, that's what we mean by 'optimal.'

For modern development organizations, however, settling for this optimal tradeoff simply isn't good enough. They want both better safety and higher velocity – at the same time.

The only way to shift the optimal velocity/safety balance point is to change the underlying assumptions that lead to the conclusion that this tradeoff is the best a team can achieve.

Specifically, the assumption that must change is the assumption that the development team should test all code before deploying it into production.

©2024 Intellyx B.V.

In other words, we've always assumed that testing in production was unsafe. Now we're saying that under the right conditions, it's safe enough.

Given today's emphasis on software velocity, we must reconsider whether it makes sense to test everything in every iteration, thus slowing down the process – or to forego testing in some situations and deploy untested code into production.

Deploying such code requires that developers carefully consider which code they should deploy without testing and how to manage the risks inherent in such a decision. There are many variables to consider: existing CI/CD processes that typically include automated testing, as well as code reviews, varied environments, and other considerations.

Once again, the challenge becomes a balancing act. How should developers prioritize which code to test vs. which code to deploy without testing it first? Given untested code is more likely to cause errors, how should developers find and fix those errors?

Rethinking Quality Assurance

To answer these questions, developers and their managers require insight into their quality assurance activities. With a tool like Faros AI, developers can gain critical insights into testing effectiveness, impact, and performance metrics that indicate how well quality assurance can impact the business while also pointing out areas of improvement.

Engineering managers can then assess various quality metrics for their teams' applications and repositories. Working with their teams, managers can help make testing more effective.

Instead of erring on either side – running too few tests thus leading to too many errors vs. running too many tests thus slowing down the development process – the right data provide the necessary insights so the development team can focus on the testing activities they should perform to maintain the optimal tradeoff between velocity and safety.

Code coverage is an important source of data for this optimization, as some code will remain untested at various times. If errors do crop up, they are more likely to come from untested than tested code.

It's important, therefore, for developers to leverage code coverage to understand which code has been partially or fully covered to avoid the same or similar errors from cropping up in the future.

Only via such careful, proactive management of untested code can development organizations shift the optimal tipping point between velocity and safety, thus improving both velocity and safety over time.

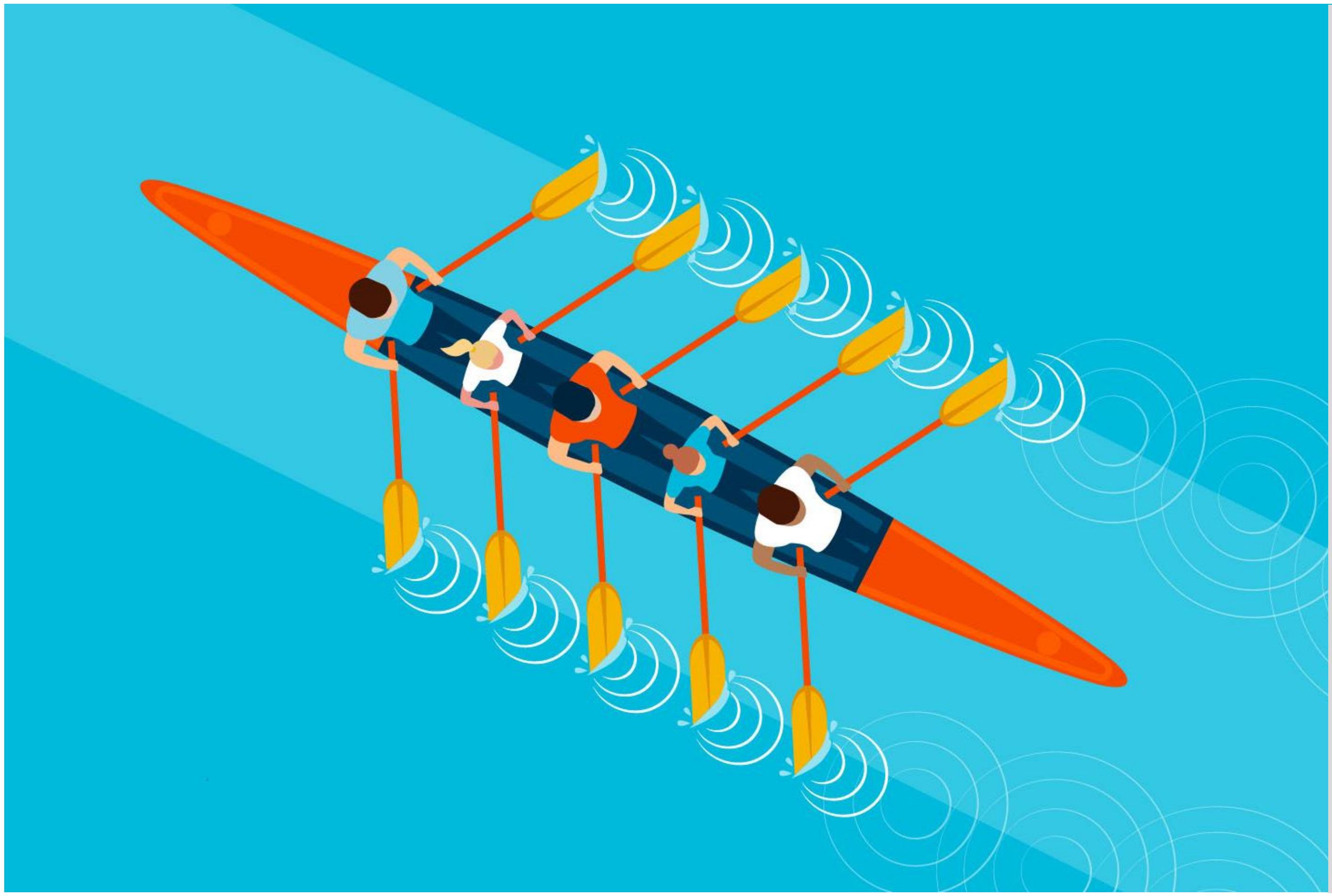
The Intellyx Take

Organizations not only tolerate issues in production, they expect them – and leverage them to deliver even greater software velocity. Today's developers are indeed following in Mario Andretti's footsteps, giving up some safety in exchange for greater velocity.

However, it is important for developers to remember Andretti's hidden message: give up too much control, and you crash and burn. Avoiding issues that adversely impact users of the software can undermine whatever competitive advantage software velocity promised.

The result is a reconsideration of the nature and importance of software quality. In the past, balancing velocity and safety has been an exercise in compromise. With insights from tools like Faros AI, development teams can rest assured they can optimize this tradeoff – without slowing themselves down.

©2024 Intellyx B.V.



By Jason English

Director & Principal Analyst
Intellyx

Achieving an Ideal Tempo with AI-augmented DevOps

Part 4 of the Mastering Modern DevOps Performance Series

In this series, we've had the opportunity to introduce the challenges of measuring developer productivity, to uncover that productivity delivers for the organization. We then explored how software development safety and velocity don't need to be at odds or create undue risk.

Still, in modern development and deployment environments, it seems like human oversight alone will never be able to get teams of developers ahead of the rate of change.

To reach our destination at high velocity, all hands on deck should not only row faster but pull in the same direction—all while aligning their efforts with a regular cadence.

The practice of AI-augmented DevOps can optimize the pace of software delivery, by measuring work outputs and correlating signals with the intentions and goals of developers and teams.

A history of misaligned incentives and goals

Remember 10–15 years ago when pundits were promoting the concept of "bi-modal IT"—in which software delivery responsibilities would be segregated into two software delivery groups working at different paces?

- **One cohort in 'fast' mode**, working in agile iterations, using the latest tools to build innovative functionality and release high-value customer-facing applications (AKA, the 'cool kids'), and;
- **Everyone else in 'slow' mode**, working to support and patch legacy apps and systems of record, which need to be slowly and carefully updated and monitored because they are too critical to fail (AKA 'the grunts').

Such pace layering represented the reality on the ground for many large enterprises. There would be one 'Innovation Team' tasked with prototyping new functionality and pushing the interface edge—totally disconnected from everyone else struggling with waterfall development dependencies, DBA requests, draconian change controls, and quarterly or annual release windows.

As analysts we relentlessly mocked bi-modal IT on several occasions. So let's not allow the advent of AI-based development tooling create another such pace separation and throw off the cadence of our organization.

Software 2.0: Developing with AI

In this prescient 2017 article, Andrej Karpathy categorizes the whole of software development as we knew it—human developers writing code without AI assistance—as ***Software 1.0.***

Thus, ***Software 2.0*** would represent the next kind of development, one where much of the work of building software is handled by intricate AI models providing coding assistance and integration help, while human "developers" aren't coding so much anymore. Instead, the '2.0 developer' identifies desirable behaviors for the system, by curating and tagging the massive machine learning datasets needed to train the AI.

Weighting parameters for AI models, instead of coding application logic, would be a new paradigm for development. However, most organizations are likely not going to be able to completely remove developer knowledge and human oversight from the logical loop.

Take Air Canada, they recently had a court order to make good on a refund offer suggested to customers by their AI-powered chatbot. Nobody was sure how the chatbot's large language model came up with the offer, but LLMs are notorious for occasionally 'hallucinating' an answer that will seem plausible or pleasing to end users.

What we really need is an AI that augments the developer's capabilities for understanding how the application they are building will fit within both integration and business contexts, so they can get into the flow of development by eliminating tedious or repetitive tasks.

Can DevEx surveys improve developer experience?

Developer surveys can be incredibly valuable in determining the quality of developer experience (or DevEx). Thought leaders at ACM recently put out an extensive study boiling down DevEx into three logical dimensions of *Flow State*, *Feedback Loops*, and *Cognitive Load*.

All three dimensions point to developers' natural desire to have engineering systems that allow them to move forward with fewer constraints, delays, and distractions. However, results of a DevEx survey are only as good as the timing of the survey, the exact wording of the questions, and the readiness of survey participants to provide accurate responses.

Time is the most constrained resource for developers. Time to finish each sprint, make that pull request, prepare a dataset, fix a hot Sev1 issue. Time to learn new skills, explore new technologies, and still have a life away from work.

No surprise, developers are unlikely to complete surveys. Further, many survey questions can deliver ambiguous conclusions from responses.

For instance, a survey might ask: "What is your satisfaction level with our current testing platform?" The organization's average response could be 3 (on a 1–5 scale).

Digging deeper into that average satisfaction level, it turns out a development team doesn't really engage with the test platform too much other than running sets of prescribed checks at each release window. If cursory tests don't fail builds very often, they might like the platform well enough, and rate it a 4 or 5.

Meanwhile, an Operations team rates the testing platform a 1 or 2, because they are dealing with resulting production failures!

Continuously measure DevEx at the source

To improve, we need to marry less cumbersome survey touchpoints with real development metrics that allow advanced algorithms to determine developer sentiment and point out morale issues.

If sentiment questions are introduced subtly, perhaps as a single thumbs-up-or-down during work, that would seem much less daunting than an extensive survey. But still, what does a thumbs-up really mean?

Non-obvious data points from the DevOps toolchain and non-verbal clues from developer actions would provide better indicators of causal patterns that represent poor DevEx, as it is concentrated down to the team and individual level.

Faros AI built a module specifically for developer experience, providing a prebuilt, curated set of data for analyzing the most relevant metrics, KPI benchmarks, activities, and events alongside survey data. For development managers and executives, this provides a great starting point for understanding the developer experience in light of system telemetry and tool usage.

Tuning a DevOps toolchain with AI provides a much faster correlation of data related to developers productively staying in a flow state, getting faster feedback loops, and having enough data and the right tools on hand to reduce cognitive load.

The correlations between surveys and telemetry increase the likelihood that future investments will deliver the desired improvements. Then, the team can set targets for DevEx success levels and identify paths forward for improvement from there, whether the development activity is coding, or tuning AI models to augment development.

Tracking toward outcomes at Coursera

Coursera grew rapidly over the last decade into one of the world's leading online learning resources. While the engineering team was busy modernizing their application estate to a more open-source-based and scalable microservices architecture, the company's culture was also heavily concerned with improving DevEx.

They established a dedicated developer productivity team to hone in on the DORA and SPACE frameworks, using platform engineering to enable new developer onboarding, end-to-end testing, and faster release cycles.

After experimenting with creating their own error-prone dashboards using Sumo Logic (a SecOps log management tool not intended for development teams), Coursera selected Faros AI to understand activity happening within several DevEx-related tools and platforms at once, from repositories to incident management to their CI/CD pipeline activity and OKR tracking.

"For measuring developer productivity, it's important to not look at just one signal but rather have a holistic view that looks at developer activity but also other important metrics like developer satisfaction and the efficiency of flow of information in the organization," said Mustafa Furniturewala, SVP of Engineering at Coursera.

.

The Intellyx Take

To survive in a software-driven world, we must constantly transform and change paradigms, or fall behind. How can we keep pace, when the rate of change is too fast for humans to comprehend?

With AI-augmented DevOps, organizations can dynamically observe developer workload and tasks, and reorder work around multiple toolsets to identify the optimal times and task assignments for more productive team design meetings, coding, and testing.

Even the best developers can leverage enhanced intelligence and timely guidance, to make the whole team better than the sum of its parts.

Copyright ©2024 Intellyx B.V.. Intellyx is solely responsible for the content of this eBook. As of the time of writing, Faros AI is an Intellyx customer. No AI chatbots were used to write this content. Image sources: Illustrations provided by Faros AI, Developer metrics from Adobe Image Create.

About the Analysts

Jason Bloomberg is founder and Managing Director of enterprise IT industry analysis firm Intellyx. He is a leading IT industry analyst, author, keynote speaker, and globally recognized expert on multiple disruptive trends in enterprise technology and digital transformation.

Mr. Bloomberg is the author or coauthor of five books, including *Low-Code for Dummies*, published in October 2019.

Jason "JE" English is Director & Principal Analyst at Intellyx. Drawing on expertise in designing, marketing and selling enterprise software and services, he is focused on covering how agile collaboration between customers, partners and employees accelerates innovation.

With more than 25 years of experience in software dev/test, cloud and supply chain companies, JE led marketing efforts for the development, testing and virtualization software company ITKO from its bootstrap startup days, through a successful acquisition by CA in 2011. Follow him on Twitter at @bluefug.

©2024 Intellyx B.V.

About Intellyx

Intellyx is the first and only industry analysis, advisory, and training firm focused on customer-driven, technology-empowered digital transformation for the enterprise. Covering every angle of enterprise IT from mainframes to cloud, process automation to artificial intelligence, our broad focus across technologies allows business executives and IT professionals to connect the dots on disruptive trends. Read and learn more at https://intellyx.com or follow them on Twitter at @intellyx.

About Faros AI

Faros AI provides innovative and reliable AI solutions giving software engineering teams and leaders much needed visibility into every aspect of the software development process — from velocity and quality, to cost, team health, organizational goals, and more. With data that is readily available and actionable, Faros AI is transforming the way engineering organizations operate, building a future where every company is a world class software company.

Learn more at https://faros.ai.