Engineering Intelligence Platform

A guiding compass for digital leaders

/thoughtworks

Strategy. Design. Engineering.

Introduction

The IT expense boom is real, with enterprise software dominating the budget. But what if you could do more with less? Enter the Engineering Intelligence Platform (EIP). This guide explores how EIPs empower developers to eliminate waste and friction in the development cycle, boosting their effectiveness and maximizing your return on investment.

In today's world, most enterprises depend heavily on software, spanning industries from automotive to oil and gas. The distinctive value offered by software has become a growth driver for many industries from banks to automobiles to airlines. Executives are not only grappling with competitive pricing dynamics but are also keenly aware of the intense market rivalry for their products. In this scenario, Information Technology (IT) organizations, particularly those specializing in software engineering, have evolved into significant cost centers for most companies. This shift has prompted executives to scrutinize the effectiveness and return on investment (ROI) of their IT organizations.

Let's start with first understanding different challenges faced by digital leaders and engineering teams in software development.

Challenges faced by engineering teams and leaders

Innovation stalled: Leaders struggle to introduce new offerings with speed and agility, potentially missing market opportunities

Scaling without sinking: Expanding operations while controlling costs proves elusive, hindering growth and profitability

Tech tangle: Fragmented IT systems riddled with redundancies create roadblocks and inefficiencies

Types of friction an engineering team goes through

Delivery friction: Not having the required infrastructure and tools for development and deployment

Access friction: Getting access to required systems to get things done

Cognitive friction: Operational complexity and large volumes of information cause cognitive overload

The entire engineering team, from leaders to developers, faces numerous challenges from scaling to dealing with frictions that must be addressed in order to meet their objectives of accelerating time-to-market while maintaining high-quality software. Additionally, there is a growing emphasis on fostering effective collaboration among engineering teams across different departments within organizations. What they seek is a highly motivated engineering team working effectively to tackle these issues within the engineering organization.

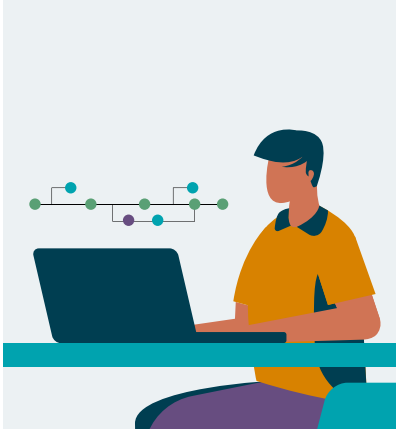
Faster time to market and releasing high quality software with confidence

Highly motivated engineering team working effectively

What is an Engineering Intelligence platform?

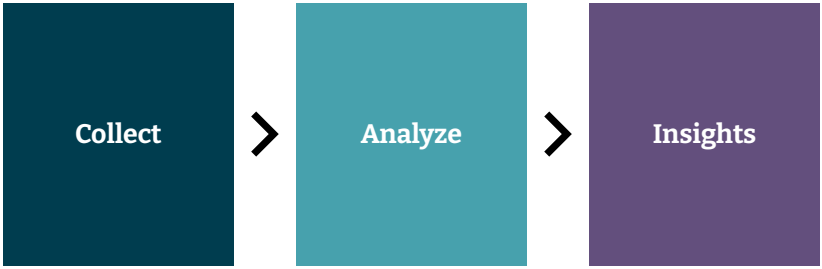
To solve challenging problems in the business domain we use a data-driven approach, or business intelligence, to analyze and understand patterns and trends to create actionable insights. The same approach can be applied to achieve engineering goals and objectives of faster time to market with releasing quality software frequently and highly motivated engineering teams working effectively. With that, the first thing to look at is what are the right metrics to measure, which will lead to achieving engineering objectives. Look at both the leading and lagging indicators which can help drive the team in the right direction.

Business intelligence



Engineering intelligence

Data driven approach to measure metrics



Adhoc, periodic or continuous

Identify patterns, analyze trends

Optimize and predict with actionable insights

Data-driven decision-making involves collecting, analyzing, and interpreting data to identify trends, patterns, and insights that can be used to make better decisions. Collecting data can happen in multiple forms: ad-hoc, such as surveys and flow analysis using value stream mapping of SDLC, periodic with retrospectives, and continuous such as collecting data from different systems, like JIRA, Jenkins (CI/CD), Github (version control), and Sonarqube (cod quality tool). Many tools available in the market seamlessly integrate with the engineering tool stack to collect real-time data. These tools analyze data over time across various projects to generate valuable insights and trends.

"Engineering Intelligence platform enables engineering teams with data-driven intelligence, leading to improved productivity, better developer experience, operational efficiency with predictable software delivery and better decisioning."

Four categories of engineering metrics

To begin with first let's look at what all engineering metrics we would like to measure and track. I recommend dividing metrics into four categories.

Delivery metrics | **Quality metrics** | **DevEx metrics** | **Operational metrics**

Delivery Metrics

These metrics revolve around understanding our performance in software delivery. In the book 'Accelerate,' the topic of delivery metrics is thoroughly explored, introducing four key metrics known as DORA 4km for measuring software delivery efficiency. Research conducted by DORA indicates that organizations excelling in these metrics are more likely to achieve success in the market. For instance, organizations boasting high deployment frequency tend to experience greater profitability and receive higher customer satisfaction ratings. The DORA 4 key metrics encompass: Deployment Frequency, Lead Time for Changes, Change Failure Rate, Mean Time to Recover (MTTR)

By focusing on these metrics, organizations can gain valuable insights into their software delivery process and enhance their overall performance.

Quality Metrics

These metrics help us to measure the quality of the software we are releasing. Releasing frequently is important, however software that is easy to use, resilient and defect-free also matters for the users of the system. Examples of good quality metrics are code complexity, build failure rate, tech debt, test coverage, security and compliance incidents, and error rate. Tech debt talked a lot about but most likely not measured and tracked due to subjectiveness. If we start recording the effort and impact of tech debt along with every tech decision as part of ADRs, we can create trends over a period of time for tech debt.

Developer Experience (DevEx) metrics

These metrics helps to understand developer effectiveness. In addition to delivering quality software to end customers on a regular basis, it is crucial for an organization to empower developers to create cutting-edge software at a sustainable pace. DevEx metrics are qualitative in nature and involve conducting surveys and retrospectives. These metrics aim to maximize developers' effectiveness by identifying and improving feedback loops and flow states, ultimately reducing overall cognitive load. Examples of DevEx metrics are related to flow state and feedback loops - onboarding time, code review cycle time, build time, service resolution time, developer happiness index.

Operational metrics

These metrics help run software efficiently in production and keep the lights on. These include metrics for System Level Indicators (SLI) which are committed to users of the system like uptime/ availability, performance, durability, incident response time also known as SRE metrics. It also includes ROI metrics such as Cloud cost per customer (as suggested in Finops) and utilization such as infrastructure and licenses

All of the above metrics in any of the 4 categories can be future looked at from the lens of leading and lagging indicators. **Leading indicators** for software development are metrics that predict future performance. They can be used to identify and address potential problems early on, before they have a negative impact on the project. Examples are Cyclomatic complexity, Test coverage, Build failure rate. **Lagging indicators** for software development are metrics that measure past performance. They can be used to track progress over time and to identify areas where improvement is needed. Examples are Time to market, Defect density, and Mean time to recover (MTTR).

Let's take an example, To improve Time to market business objective, first metric to start measuring and tracking is Deployment frequency as lagging delivery metric, with objective to improve it from quarterly releases to every sprint releases, couple of ways to improve deployment frequency is by 1) good automated test coverage which allows to reduce testing cycle 2) quick turn around on security and compliance cycles for releases... So we need to start measuring "Testing cycle time" and "Security & compliance cycle time" as leading metrics and start implementing automation in functional, security and compliance testing which helps improve Deployment frequency (lagging indicator) and eventually Time to market for product.

"Monitor lagging metrics you want to change and identify leading metrics that the team should focus on to action that change."

Timothy Cochran, Technical Director, Thoughtworks

Example of metrics to measure and track across four categories with business outcomes

	Business outcomes	Metrics to measure
Delivery metrics	Improve time to market Improve customer NPS Better predictability	DORA 4 key metrics: Change lead time, Deployment frequency, Mean time to recover, Change fail percentage
Quality metrics	Reduce cost Better predictability Improve resiliency Improve customer NPS	Code complexity, Build failure rate, Tech debt, Test coverage, Security and compliance incidents, Production issues
DevEx metrics	Employee retention Improve developer NPS Better predictability Improve time to market	Onboarding time, Code review cycle time, Commit to deploy time (SAST & DAST checks, tests, build & deploy time), Ticket resolution time, Developer happiness index (Dev NPS)
Operational metrics	Reduce cost Improve resiliency	SRE metrics – SLA, SLO, SLI, Cloud cost per customer Resource utilization (infrastructure and licenses)

The above is a suggested list and not exhaustive. Choose ideally 3 and max upto 5 metrics in each category with clear engineering goals in mind. Drive metric improvement through actionable measures, linked to clear ways to confirm whether engineering goals have been achieved.

"Measure what matters. Choose metrics to measure with clear engineering goals and objectives."

Here are some tips for using metrics effectively:

- Choose metrics that are specific, measurable, achievable, relevant, and long term.
- Don't overload a single metric with multiple purposes.
- Set realistic goals for each metric with clear engineering objectives leading to business outcomes.
- Track your progress over time and make adjustments as needed.
- Use metrics to identify waste and find areas for improvement. Don't use them to punish or judge people.

Establishing a line of sight to business outcomes

One of the recurring challenges faced by engineering leaders is bridging the gap between high-performing engineering metrics and desired business outcomes. EEBO (Engineering Excellence to Business Outcomes) Metrics offer a framework for a systematic approach to establishing a correlation between engineering excellence and business outcomes. EEBO Metrics provides a method to draw statistical correlations between metrics that reflect excellence in software and those indicating superior production deployment, with progress toward achieving desired business outcomes.

Structure Of EEBO Metrics

Metrics reflecting excellence in software development

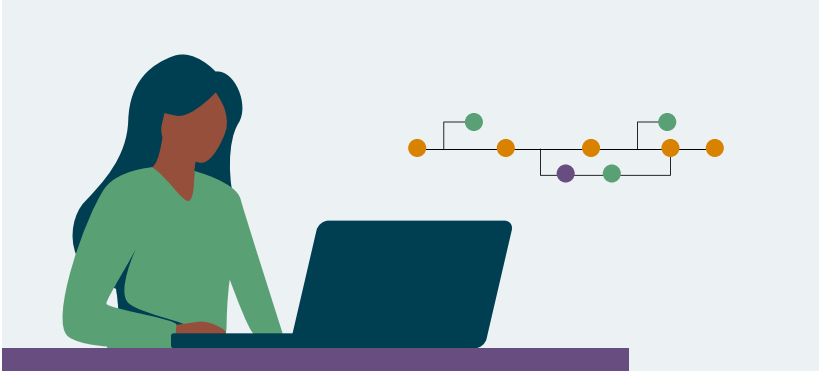
Metrics reflecting excellence in production deployment

Metrics categories reflecting progress towards desired business outcomes

EEBO Metrics essentially measure a team's deviations from their agreed-upon goals over time. This trend is then checked for correction with the trend of business outcomes to establish the presence or absence of correlation. Absence or negative correlation may indicate misalignment or wrong prioritizations for which the framework proposes next best actions.

Let's explore a few examples to understand how we can collect, measure and track metrics linking to engineering objectives and eventually business outcomes.

a) Analyzing git commit logs to identify patterns for best practices as well as waste in the software development



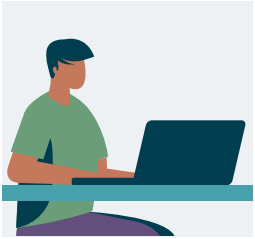
Git commit logs can be a valuable source of information for analyzing software development practices and identifying waste. By analyzing these logs, teams can identify patterns in their development process that can lead to improvements in efficiency and quality.

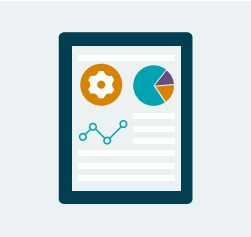
Best practices in software development include practices like making small and frequent commits, while wasteful or anti-pattern practices include maintaining long-lived branches and enduring lengthy pull-request approval cycles.

To analyze your team's development process, key metrics derived from git commit logs are invaluable. These metrics encompass factors such as commit size, commit frequency, and code review time. By consistently monitoring and analyzing these metrics over time, you can pinpoint trends within your team's development workflow. This enables you to recognize areas of improvement as well as potential regressions.

Commit-driven metrics fuel efficient coding practices, enhancing code quality, developer experience, and ultimately, customer satisfaction.

b) Analyze Service request ticket data to identify patterns for service requests and opportunities for self-service



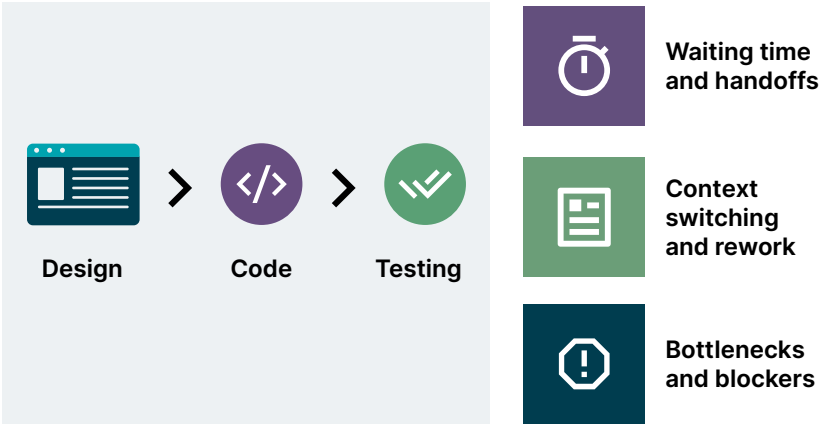


By analyzing IT tickets system data of engineering team requests such as creating git repo or network firewall changes etc, you can identify patterns for service requests and areas of improvement. This information can be used to improve the efficiency and quality of your service delivery. Some common metrics include the most common service requests, and average time to resolution.

Through this analysis of service tickets, we can identify which service requests are suitable for self-service. Implementing self-service options not only enhances user experience and expedites task completion but also reduces costs and boosts developer efficiency.

There are a number of different ways to transfer service requests to self-service. One common approach is to create a knowledge base that contains articles and tutorials that customers can use to resolve common issues. Another approach is to develop self-service tools, such as chatbots or virtual assistants, that can help customers to resolve issues without having to contact a human support representative. By transferring these types of service requests to self-service platforms, you can free up your support team to focus on more complex issues and improve the overall customer experience.

c) Flow analysis using value stream mapping (VSM) to identify bottlenecks and eliminate waste in SDLC



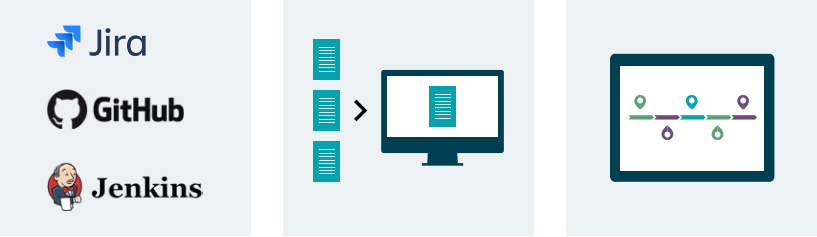
Value stream mapping (VSM) is a lean technique that can be used to visualize and improve the flow of materials and information in any process. In the context of software development, VSM can be used to map the entire software development life cycle (SDLC), from requirements gathering to deployment. VSM can help software development teams to identify and eliminate waste, which is anything that does not add value to the customer.

In the context of software development, waste can include things like rework, waiting time, and handoffs. VSM can help teams identify and eliminate waste by visualizing the entire SDLC and identifying areas where there are bottlenecks or inefficiencies, such as a long queue for code reviews or a slow build & deployment process. Once the bottlenecks have been identified, the team can take steps to address them, such as pair programming as a continuous review process or automating the deployment process.

Using VSM, we can capture metrics, such as timing for each step in the software development flow, and track cycle times over time.

By implementing improvements focused on waste reduction and bottleneck elimination, we aim to enhance overall efficiency.

d) Predicting delivery timeline with data from JIRA, Git repository and CI/CD



Let's look at some futuristic use cases of the engineering platform. Predicting software delivery time is challenging. Once we start capturing historical data from Jira, Git repository, and CI/CD, we can use a variety of machine learning and statistical techniques to predict delivery timeline with data. The approach is an extension of the "Yesterday's weather" technique, with large amounts of data.

One approach is to use a supervised learning technique, such as a regression model. To train the model, you would need to collect historical data on delivery timelines, as well as data on the features that you believe influence delivery timelines, such as the number of Jira tickets, the size of the Git repository, and the number of CI/CD builds. Once the model is trained, you can use it to predict delivery timelines for new projects or features.

Another approach is to use an unsupervised learning technique, such as clustering. This approach can be used to identify groups of projects with similar delivery timelines. Once the projects have been clustered, you can use the average delivery timeline for each cluster to predict the delivery timeline for new projects.

Finally, we can also use a combination of supervised and unsupervised learning techniques. For example, you could use a clustering algorithm

to identify groups of projects with similar delivery timelines, and then use a regression model to predict the delivery timeline for each cluster.

Predicting software delivery is based on many contextual parameters such as project complexity, organization ecosystem, team capability, etc. and needs to be applied with care and not generalized too much.

Now that we've gained insights into what to measure, how to measure it effectively, and various tracking techniques, let's explore the available tools for implementation.

Tools available in the market

In this space, numerous players offer a wide array of capabilities focused on continuous data collection,real time analysis and the provision of insights & trends including alerts for threshold crossings. Many of these tools integrate seamlessly with your existing engineering toolstack. While it is difficult to find one single tool that covers all engineering metrics,, the space is maturing and tools are improving with every new release. It's crucial not to begin by selecting tools and then determining what to measure. Instead, start with identifying objectives and problems to solve.Then, decide what metrics to measure, focusing on areas for improvement. Afterwards, choose the appropriate tool that aligns with your objectives.

Faros.AI empowers you to measure performance across various lenses, providing a holistic view. Whereas, DevEx 360 stands out for measuring Developer experience and Code Climate is leading in code quality metrics.

FAROS · PLURALSIGHT FLOW · swarmia · DX · SLEUTH

CodeScene · Allstacks · waydev · CODE CLIMATE · Athenian

Hatica · JELLYFISH · LinearB · OOBEYA · Acumen.io

Disclaimer: Please note that the list of tools provided above is the result of a brief research effort aimed at illustrating the diverse range of options available, not a personal recommendation. Additionally, comparing these tools can be challenging due to the continual evolution of their features.

To look for tools available in the market, you can use the search terms, “Engineering intelligence platform”, “Software Development Analytics platform”, and/or “Software Delivery intelligence platform” You can also find a list of tools curated on the Gartner website here.

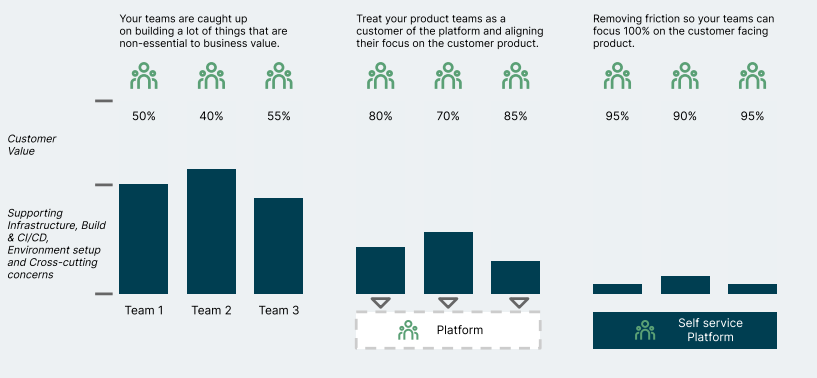
At Thoughtworks, we have built our own internal tool called “Polaris” for measuring engineering effectiveness and helping our customers by bringing insights and showcasing how engineering practices help to achieve business goals.

Polaris

Engineering platform to maximize developer effectiveness

An engineering platform minimizes friction by simplifying and concealing non-essential complexities that do not directly contribute to business value. Utilizing an engineering platform, particularly one equipped with a self-serving internal developer platform, is crucial for reducing obstacles and optimizing developer productivity.

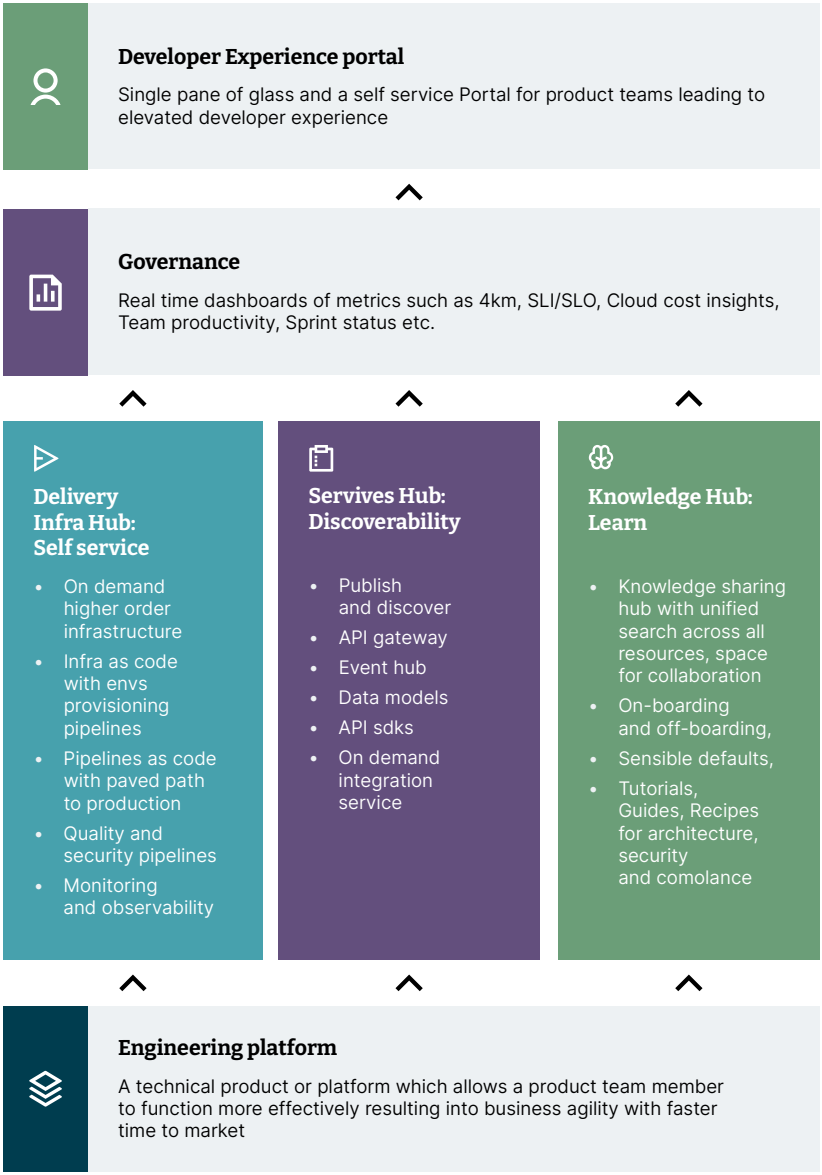
Although the sources of friction vary for an engineering team, they can all be characterized in the same way; they require time and energy but do not create end-user value. For instance, product teams might find themselves engaged in tasks such as supporting infrastructure or configuring CI/CD pipelines/environments.While these activities are required, they don't contribute directly to customer value. Excessive non-value adding tasks result in frequent context-switching, which significantly increases the cognitive load on developers. This reduces developers' ability to achieve flow and maximize value delivery.



How can an engineering platform help?

In general, platforms are technology foundations that accelerate an enterprise's ability to deliver customer value. An engineering platform does the same thing for its software development teams, who are, in effect, its customers. A good engineering platform consists of three core components: a delivery infrastructure hub, a service hub and a knowledge hub. The platform should be supported by lightweight governance and a frictionless developer experience portal.

Building blocks of an engineering platform



The Delivery Infrastructure (DI) hub

The Delivery Infrastructure (DI) hub provides product teams with self-service and automated 'golden paths' that go from development to production with quality and security checks baked in. The hub includes common cross-platform tools such as continuous integration setup, observability and monitoring capabilities. It also accelerates building higher-order infrastructure products and environments using Infrastructure-As-Code (IaC) on top of existing cloud services.

The services hub

The services hub offers an on-demand ability to publish and discover APIs and events along with documentation and Software Development Kits (SDKs). It helps product teams reuse and compose digital assets which limits dependencies and accelerates delivery.

The knowledge hub

The knowledge hub breaks organizational silos by making it easy to publish project documentation, org-level tutorials, guides, recipes for on-boarding/off-boarding, architecture, sensible defaults and security and compliance information and policies. The unified search also helps developers discover solutions faster. With proper documentation and automation, we have seen onboarding times reduced from months/weeks to days.

Automated governance

Automated governance leveraging observability stack provides real-time dashboards and insights of operational metrics that are related to software delivery and production systems such as DORA four key metrics (4KM), availability, cloud cost alongside continuous monitoring and tracking of metrics and compliance requirements. Gartner breaks it down into two categories – the engineering intelligence platform and the engineering value stream management platform.

A developer experience portal

A developer experience portal provides a single point for self-service capabilities for product teams, leading to elevated developer experience. For example, Spotify developed an open-source platform for building developer portals called Backstage. While it provides out-of-the-box core features, organizations still need to tailor it to their needs. We used Backstage to build a portal for TELUS called Simplify, improving developer experience for their 8,000 engineers. Learn more about that, here.

At Thoughtworks, we built our own award winning engineering platform called NEO to enable effective software development. NEO has sped up the application development process, from idea to outcome by an average of 30%.

An engineering platform helps with improving Developer experience and Delivery metrics by reducing friction and associated waste in the system eventually leading to happy and motivated engineering teams.

Gotchas and anti-patterns

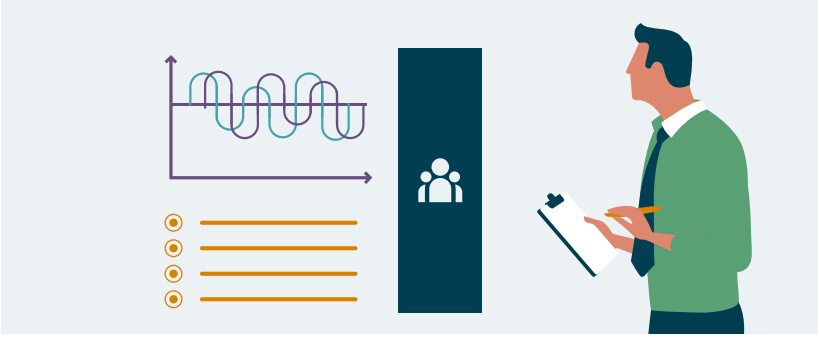
Metrics are a double edged sword. Executives often rely on numbers as the primary means of gauging progress. While metrics, when employed with good intentions, can assist in goal attainment, they can also be easily manipulated when used to evaluate individuals and their performance.Hence, it's essential to be wary of pitfalls and negative patterns associated with excessive reliance on metrics. Additionally, assigning multiple purposes to a single metric can lead to numerous complications.

Measure and collect metrics at the team level and not at the individual level

Measuring metrics at an individual level might not help to achieve the overarching goals we have set. For example, a senior developer may be more involved in reviewing pull requests and helping other team members resolve blockers and dependencies.

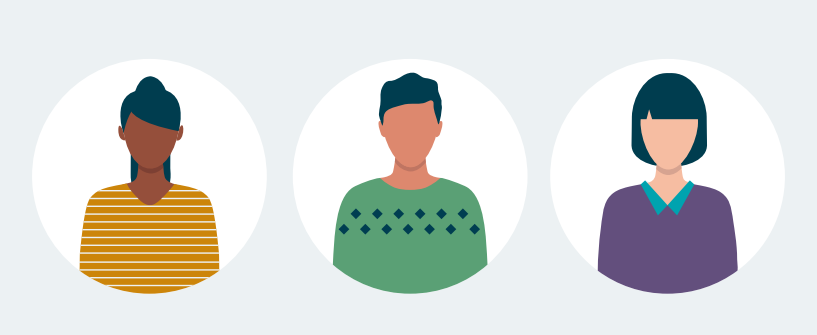
Whilst their individual metrics may be impacted this, the benefit to the team, and collective goal, is high.

Focus on trend analysis within a specific team over time, rather than comparing across different teams



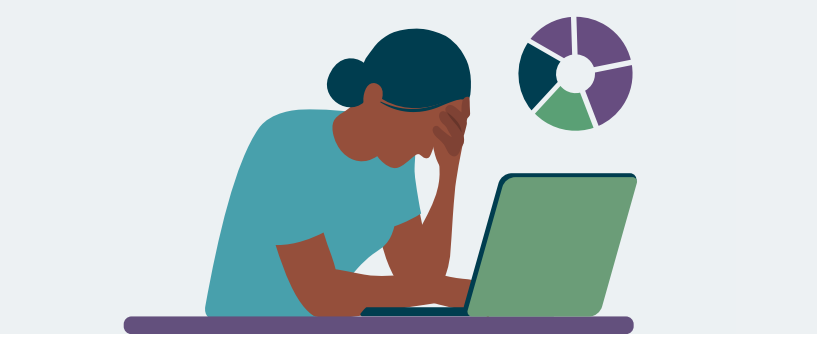
Comparing metrics across teams can be counterproductive.
For instance, if we compare the deployment frequency of the “WebApp” and “MobileApp” teams, it’s not always meaningful. The WebApp team might deploy multiple times a day, while the MobileApp team’s deployments are constrained by processes like the AppStore approval cycle. Similarly, comparing metrics between teams working on different types of projects, such as core banking software versus marketing corporate websites, can lead to misleading conclusions.

Do not use metrics to judge individuals or teams performance



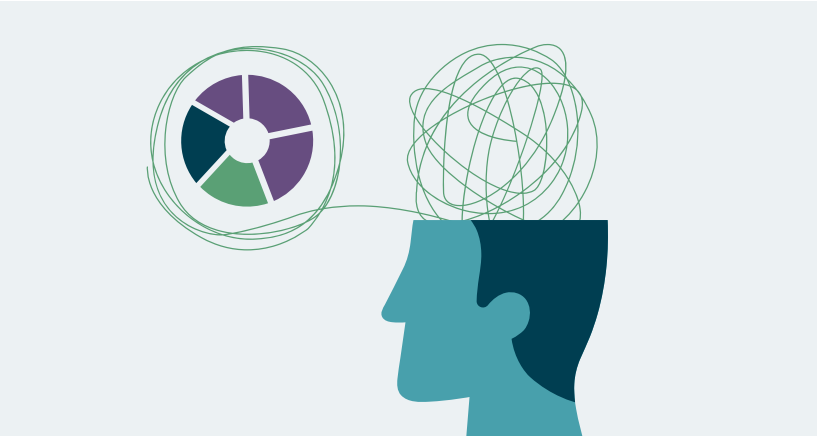
While metrics can be a valuable tool for tracking progress and identifying areas for improvement, it is important to use them carefully and avoid using them to judge individuals or teams. Doing so can lead to a culture of fear and blame, people focusing on the wrong things, can be unfair. E.g Individuals can start focusing on their performance over team outcomes, eventually leading to a blame game. Use metrics to identify areas where people need help. Don't use them to punish people.

Overloading a single metric with multiple purposes causes many problems



While it may be tempting to use a single metric to simplify reporting or to make it easier to compare performance across different teams or products, this can lead to a number of problems. It can be confusing and lead to suboptimal decision-making. Choose metrics that are specific, measurable, achievable, relevant, and time-bound.

Collecting metrics should not become a burden to the team



While metrics serve as valuable instruments for monitoring progress and pinpointing areas in need of enhancement, it's crucial to exercise caution against excessive metric collection. Overloading the team with too many metrics can impose a burden, wasting time and resources, and potentially leading to "paralysis by analysis," ultimately demoralizing the team. It's advisable to select metrics aligned with engineering goals and, where possible, automate their collection and analysis processes.

Resources

Sunit Parekh's Talk at XConf India 2023 in Hyderabad:
Engineering intelligence, data-driven visibility into the engineering effectiveness by Sunit Parekh

To learn more about engineering intelligence and metrics to measure, here is a list of articles for your reference:

- An Appropriate Use of Metrics by Patrick Kua
- Maximizing Developer Effectiveness by Tim Cochran
- Cannot Measure Productivity by Martin Fowler
- Engineering platform: key to maximizing software development effectiveness by Sunit Parekh
- DevEx: What Actually Drives Productivity on ACM
- Outcome Over Output by Martin Fowler
- The Worst Programmer I Know by Dan North
- Measuring Developer Productivity via Humans by Tim Cochran & Abi Noda

Author

Sunit Parekh
Head of Digital
Platforms Practice, India

With over 20 years of experience, I'm a seasoned technology strategist passionate about helping clients achieve their digital goals. I specialize in guiding large enterprises through complex distributed projects, from global solutions to digital transformations. My expertise lies in crafting impactful technology strategies and implementing cutting-edge cloud-native solutions across ambitious projects.

Modern Engineering Advocate and Cloud-Native Champion
I'm a firm believer in leveraging the power of cloud ecosystems and embracing cloud-native approaches to build modern, scalable infrastructure. I'm equally passionate about collaborating with clients who share my commitment to adopting modern engineering practices for achieving technical excellence.

Open Source Contributor
Beyond my client work, I actively contribute to the open-source community. I've built a valuable tool, Data Anonymization, that helps developers safely anonymize production data for testing purposes.

Thoughtworks is a global technology consultancy that integrates strategy, design and engineering to drive digital innovation. We are over 11,500 Thoughtworkers strong across 51 offices in 18 countries. For 30 years, we've delivered extraordinary impact together with our clients by helping them solve complex business problems with technology as the differentiator.

thoughtworks.com

/thoughtworks

Strategy. Design. Engineering.