



AHEAD

Enhancing Development Efficiency with Internal Developer Platforms

Companies in today's rapidly evolving software development landscape are always looking for methods to increase output, simplify procedures, and speed up the delivery of high-quality software. Implementing **internal developer platforms (IDPs)** and investing in platform engineering are potent methods for reaching these objectives. These creative approaches aim to equip developers with the tools and frameworks necessary to execute on expected timelines and deliver desired outcomes. According to Gartner, these portals allow software engineering executives to "establish a flexible 'app store' that enhances software reuse, elevates the experience of onboarding developers, simplifies software delivery, and promotes knowledge sharing."

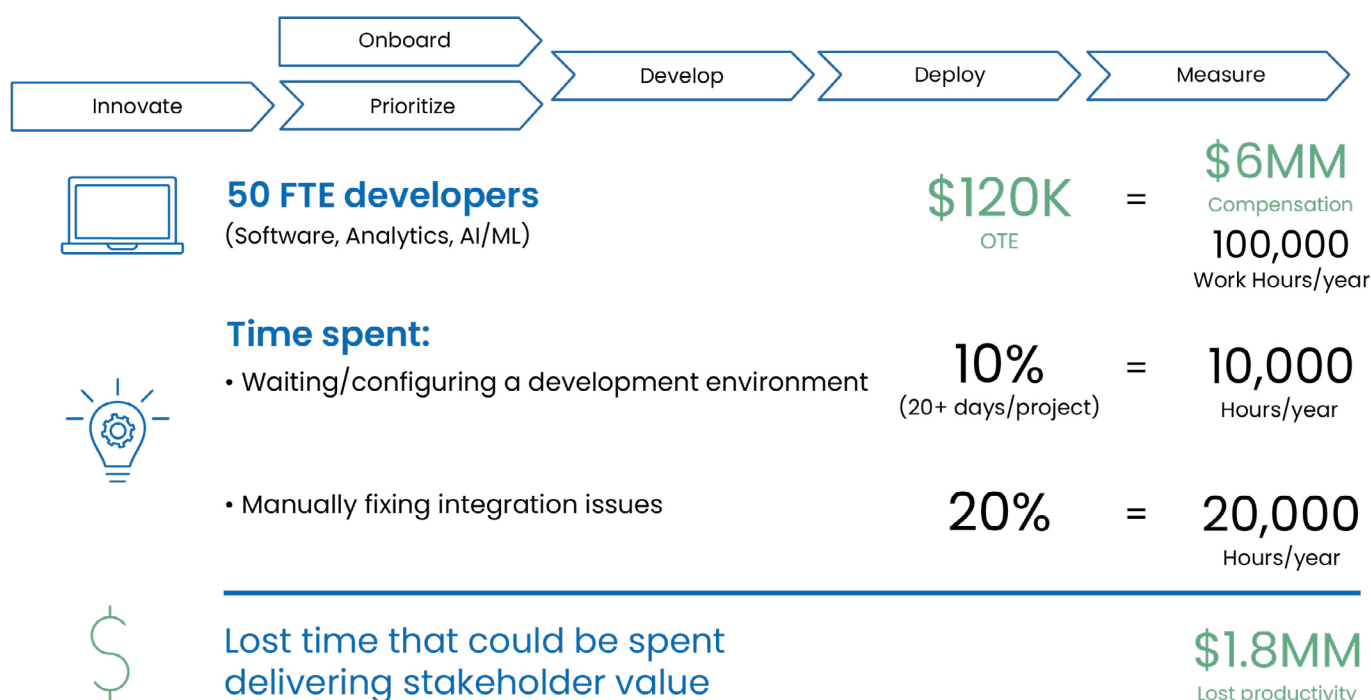
In this whitepaper, we'll discuss what internal developer platforms are, why they're so important, and how platform engineering can help organizations build top-notch dev teams.

What are Internal Developer Platforms?

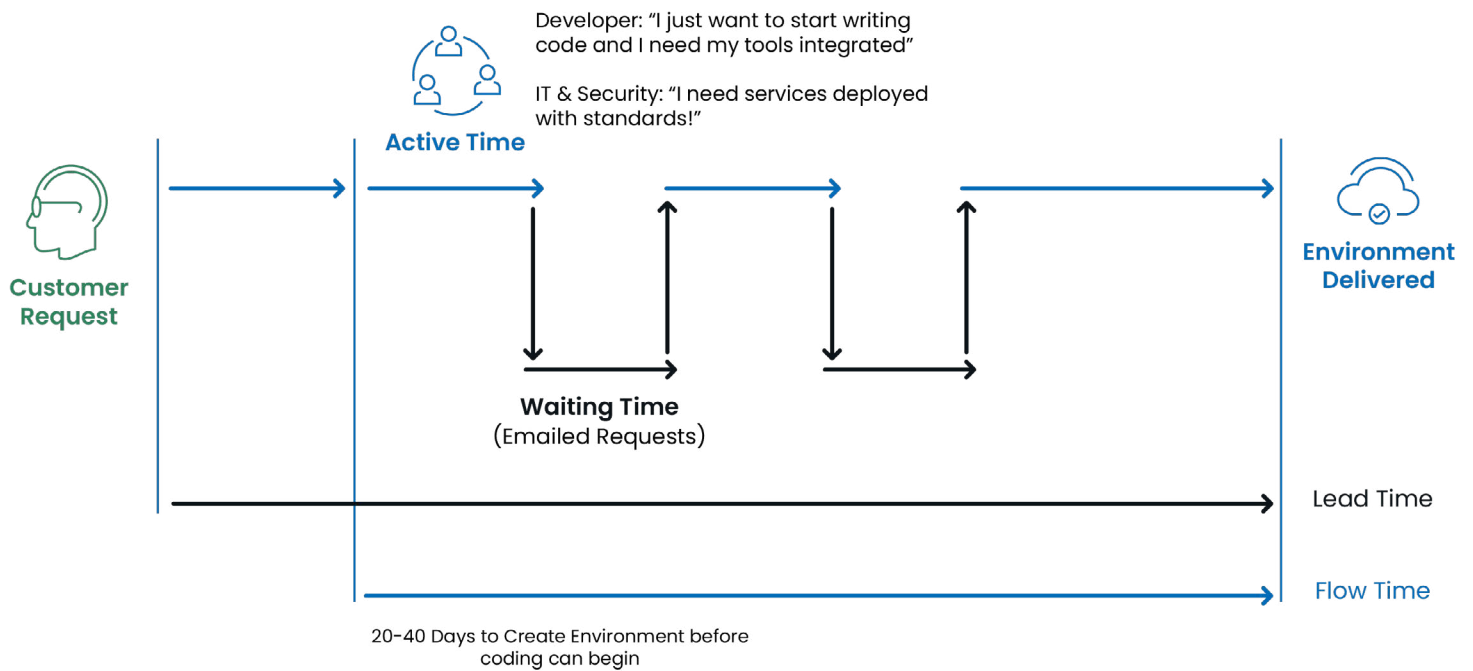
Put simply, an IDP is an in-house software acceleration platform that streamlines and automates the development process to boost efficiency and creativity. IDPs are becoming increasingly relevant, with [Gartner](#) predicting that by 2026, 80% of software engineering companies will have created platform teams. But what makes them so necessary?

Ideally, developers wouldn't have to worry about the intricacies of infrastructure management, implementation, or the integration of numerous resources and applications, and would instead be able to focus on creating code and adding features. Software developers are some of the most expensive assets that a technology department has, and the cost of environment delays and incomplete deployments add up. The mitigation of these challenges and costs is the ultimate goal of internal developer platforms.

THE PAIN IN THE DEVELOPMENT PROCESS ADDS UP



COMPLEXITY & RECURRING INFRASTRUCTURE PROBLEMS



Key Components of an Internal Developer Platform

Client-Initiated Interactions

Without the need for assistance from other teams, developer teams can self-serve environments with IDPs. IDPs provide straightforward interfaces that developers can utilize to provide assets, deploy applications, and oversee infrastructure automatically.

Reusable Components

Developers can use established standards and patterns through the tool's library of reusable elements and templates.

Automated CI/CD Pipelines

These systems use automated CI/CD pipelines to speed up the software development lifecycle from creation to release.

Monitoring & Observability

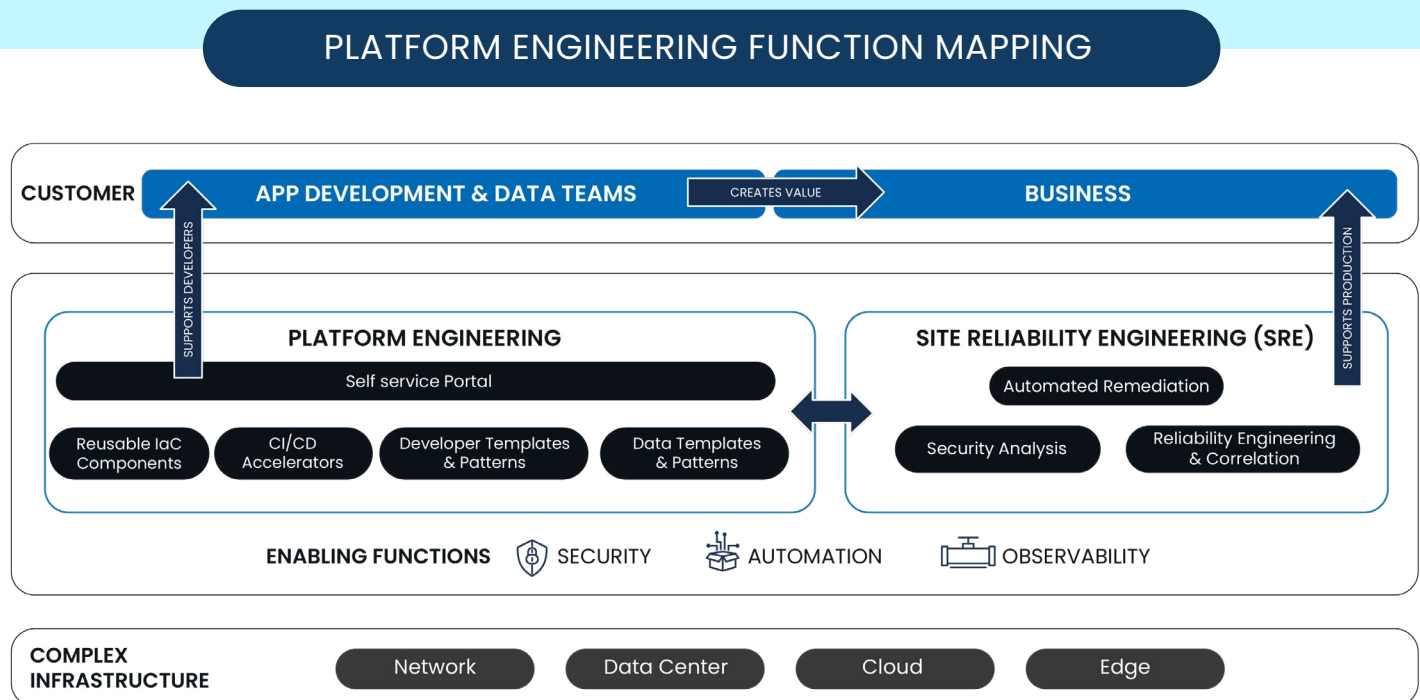
IDPs typically come with monitoring and observability tools to aid developers in finding and fixing bugs in their software. Baseline MELT (metrics, events, logs, and traces) figures enable site reliability engineering after deployment and are typically boilerplate.

Everything as Code

Developers can now specify their [infrastructure as code](#) and maintain it with version control because of IDPs' promotion of the 'everything as code' mentality, observability, policy, and governance.

The Role of Platform Engineering

Platform engineering refers to creating and maintaining internal platforms for software developers. Making these platforms run well necessitates developing consistent tools, procedures, and guidelines. The power of IDPs can be unlocked in the following ways thanks to platform engineering:



Designing User-Centric Experiences

Platform engineers frequently elicit feedback from developers to learn about their wants and challenges and identify areas for improvement. They create IDPs with the digital producer in mind, making it simple for programmers to find the necessary tools and navigate the interface.

Commoditizing Proven Methods

Engineers on the platform set standards for code quality, safety, and regulatory conformance. To ensure that developers follow these standards, they design templates, guidelines, and boilerplates which are supported and maintained for ease of use and consumption.

Ensuring Reliability & Scalability

A solid IDP should be dependable and extensible to accommodate an expanding development staff. Platform engineers oversee the infrastructure's design and upkeep, ensuring it can scale with demand without compromising availability.

Monitoring & Continuous Improvement

Platform engineers implement monitoring and observability solutions to keep tabs on how well IDPs and the applications they host are doing. This information is then used to pinpoint problem areas and inform future platform updates.

Why Internal Developer Platforms Matter

Now that we've discussed the role of IDPs and platform engineering, we can explore why they are crucial in modern software development by examining their inherent advantages:

Increasing the Pace of Development

IDPs simplify the development process, allowing programmers to spend more time creating code and less time tinkering with the infrastructure and deployment processes. Time-to-market and feature delivery speed are both improved as a consequence.

Improvements in Teamwork

Collaboration across departments is facilitated through IDPs. There is less friction and faster project delivery when developers, operational teams, and QA engineers work together in a uniform environment.

Boosting Trustworthiness

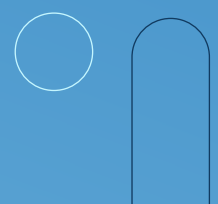
IDPs are created with dependability in mind as a result of platform engineering. There is less room for application mistakes when automated procedures and best practices are implemented.

Enabling Scalability

Scalable infrastructure and development procedures are in high demand as businesses expand. Integrating data platforms and platform engineering lays the groundwork for flexible resource and application scalability.

Attracting & Retaining Talent

Companies that invest in their developers with cutting-edge workspaces and tools are more likely to keep them around. Including IDPs in recruitment and employee retention strategies can be a competitive advantage.



Implementing Internal Developer Platforms & Leveraging Platform Engineering

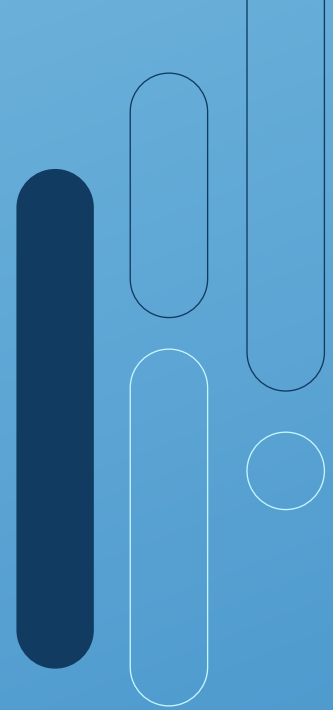
Here are five keys to success for organizations to prioritize as they consider adopting IDPs and platform engineering:

1. Understand Your Developer Needs

Get to know the development teams' requirements and problems first. Gather information that will be used to develop your IDP through surveys, interviews, and workshops. This holistic method helps you better understand their habits and gets them involved and invested in the process. By depicting common processes, user journey mapping helps identify bottlenecks where the IDP can have the greatest impact. Further, maintaining the IDP's flexibility and responsiveness to new demands can be accomplished by establishing ongoing feedback loops.

2. Foster a Culture of Collaboration

Inspire developer teams and platform engineering groups to work together! When developers have input into the iterative development process from the start, the resulting product is more likely to fit seamlessly into their existing processes. Sharing information between developers and platform engineers benefits everyone involved – developers learn about the nuances of the underlying infrastructure, while platform engineers gain an appreciation for the difficulties that developers encounter. Collaboration of this sort lays the groundwork for an IDP that quickly responds to changing conditions and trends in the market.



3. Prioritize Automation

Make automatic code deployment and infrastructure provisioning standard as part of your development cycle. In doing so, errors are minimized, and progress is sped up via automation. The IDP represents a catalyst for creativity and effectiveness due to this culture of automation, which not only speeds up development but encourages a culture of experimentation and constant improvement.

4. Invest in Training

Give developers the tools they need to succeed with the IDP, including training and documentation. Developers that feel they have control over the platform are more inclined to use it. Thorough instructions, frequently asked questions, and advice for fixing common issues should all be included in the documentation. Providing developers with these tools helps them become more technically proficient and makes them feel more ‘in charge’ of the platform. When developers are confident in their available resources, they are more likely to actively adopt the IDP and seamlessly incorporate it into their existing processes. Your dedication to their personal and professional development will be reflected in this investment in training and assistance, which will help create a more upbeat and cooperative atmosphere at work.

5. Continuously Improve

The sustained success of an IDP requires more than just a one-time effort. Constantly reassessing its usefulness will keep it in step with your development teams’ changing requirements. Incorporate data, feedback loops, and user interaction to better understand how the IDP functions in actual development. Your IDP can continue to be useful and even serve as a catalyst for new ideas and approaches to software development if you foster a culture of continual improvement throughout the organization.



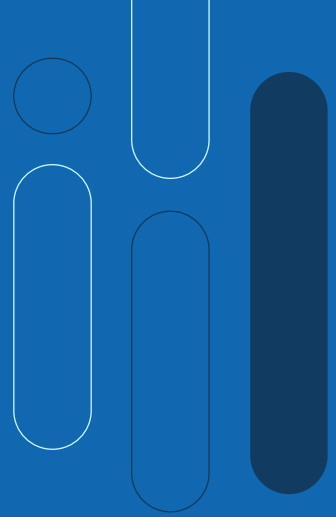
Achieving Measurable, Outcome-centric IDP Implementations

Measuring the success of internal development platforms is essential to assess their impact on the organization and identify areas for improvement. The specific metrics you choose to measure may vary depending on your organization's goals and the nature of your software development projects. Here are some common measurements for success related to internal development platforms:

- **Time-to-Market:** Measure the time it takes to develop and release software features or products. A successful IDP should help reduce development cycles, enabling faster delivery to customers.
- **Feature Deployment Frequency:** Track how often new features or updates are deployed to production. A higher deployment frequency indicates that development teams are using the IDP efficiently to deliver value to users more often.
- **Code Quality:** Monitor code quality metrics, such as code review completion rates, code coverage, and static code analysis results. Improved code quality can lead to fewer bugs and faster development.
- **Deployment Success Rate:** Measure the percentage of deployments that are successful without causing downtime or errors. A fully functional IDP should help ensure reliable deployments.
- **Lead Time for Changes:** Calculate the time it takes for a code change to go from idea to production. Shorter lead times indicate that development processes are more efficient. Being able to identify and measure bottlenecks in the development process, such as long code review times or delays in infrastructure provisioning, can lead to faster development and improved efficiency.
- **Developer Productivity:** Assess developer productivity by tracking metrics like lines of code written, bugs fixed, and tasks completed per developer. An increase in developer productivity may indicate the IDP's effectiveness.

- **Onboarding Rate:** Measure the time it takes to onboard new developers and get them productive within the IDP. A shorter onboarding time can lead to faster team integration.
- **Collaboration Metrics:** Measure collaboration and knowledge sharing within development teams. This can include metrics like the number of contributions to shared documentation or the frequency of cross-team communication. Track the creation and utilization of documentation, tutorials, and knowledge-sharing resources within the IDP. Effective documentation can improve efficiency, reduce the learning curve for developers, and improve your internal and external collaboration.
- **Security and Compliance:** Evaluate the IDP's impact on security and compliance by tracking metrics such as the number of security vulnerabilities detected and resolved or the adherence to compliance standards. Monitor how well development teams adhere to internal coding standards, best practices, and compliance requirements. Improved adherence can lead to fewer issues and faster development.
- **Innovation Metrics:** Track the number of innovative ideas or experiments initiated by development teams using the IDP. Successful IDPs should encourage creativity and innovation.
- **Error and Incident Rates:** Measure the frequency and severity of errors, incidents, or outages in production. A successful IDP should help reduce the occurrence and impact of such issues.

Remember that the specific metrics you choose should align with your organization's objectives and the goals you set for the IDP implementation. Regularly reviewing these metrics and making adjustments based on the insights gained can help ensure the ongoing success of your internal development platform and how it is being utilized.



Conclusion

The adoption of platform engineering and internal developer platforms can radically improve the efficiency of development teams. Organizations can improve their development pace, dependability, and capacity to recruit top personnel by streamlining workflows, establishing best practices, and encouraging collaboration.

Keep in mind that the success of your IDP and platform engineering investments hinges on matching technology with the requirements of your developers as you set out on this path. Empowering your development teams to succeed in the dynamic field of software creation is possible with the correct instruments, procedures, and a dedication to constant enhancement.

Contributing Author:

Stephen King, Chief Architect

AHEAD

Combining cloud-native capabilities in software and data engineering with an unparalleled track record of modernizing infrastructure, we're uniquely positioned to help accelerate the promise of digital transformation.

National Hubs

ATLANTA

1117 Perimeter Center
W406
Atlanta, GA 30338

CHICAGO

401 Michigan Ave.
#3400
Chicago, IL 60611

SAN FRANCISCO

2000 Crow Canyon Place
Suite 250
San Ramon, CA 94583