WHITE PAPER

PLATFORM ENGINEERING

Abstract

Platform Engineering (PE) is an emerging discipline that focuses on designing and building toolchains and workflows that enable self-service capabilities for software development teams who are facing the challenges presented by modern software development in the cloud-native era. The main objective of PE is to give development teams liberty to focus on developing the code while consuming capabilities provided by PE such as the Continuous Integration/ Continuous Delivery/Continuous Testing (CI/CD/CT) as a service. This point of view first describes these challenges as the forces behind PE and then discusses the PE approach to address them. It then provides a view of the different solution capabilities that are emerging in the market. Finally, it provides the future trend as to how this discipline is going to mature and the role Infosys can play in successfully realizing the objectives of PE.

Infosys®
Navigate your next

Why Platform Engineering?

Over the last 20 years as software has taken center stage in running businesses smartly and effectively. Many new trends in its development have emerged starting from single tier to multi-tier and now cloud native. The efficiency and automation in development lifecycle have increased the reliability and quality of the software applications. Some of these trends, which made the most profound impact in the recent years, are around increasing the speed with which software features are delivered by a product-centric development methodology based on the practice of Agile as well as by the automation of software deployment through the practice of DevOps (or its successor, DevSecOps). In parallel, as the various public cloud offerings have evolved, they have rolled out new capabilities and services for developing and deploying applications. A hybrid cloud computing approach in which applications are hosted in a combination of different cloud and on-premises environments, is increasingly being adapted. The multi cloud ecosystems have increased the deployment complexity for software developers.

In a perfect DevSecOps world, software developers pivoted from the traditional "throw it over the wall to operations" model to deploying and running their applications and services end to end. As Amazon's CTO Werner Vogel described in 2006 – "You build it, you run it." Additionally, Google popularized the notion of Site Reliability Engineering (SRE) that ensures that the software running in production is reliable, highly available, and performant. Site Reliability Engineers are responsible for the availability, latency, performance, efficiency change management, monitoring, emergency response and capacity planning of their services. They use service-level objectives (SLOs) and error budgets to set shared expectations for performance and balance reliability with innovation, respectively.

However, the benefits of the above changes have not been realized uniformly across all organizations. As documented in the blog "What Team Structure Is Right for DevOps to Flourish?" by Mathew Skelton as one of the anti-patterns, when an organization tries to implement true DevOps and removes dedicated operations roles, developers become responsible for infrastructure, managing environments, monitoring etc., in addition to their previous workload. Often senior developers bear the brunt of this shift, either by doing the work by themselves or by assisting their junior colleagues. This type of antipatterns has been clearly visible in several studies, including State of DevOps by Puppet and Humanitec's Benchmark study. The latter clustered top and low performing organizations, based on standard DevOps metrics (lead time, deployment frequency, mean time to recovery and defect injection ratio) and a stunning 44% of low performing organizations experience the mentioned antipatterns – where their most valuable development resources are not focused on developing software features. Compared to that, the top performers (3.1% of the organizations) successfully implemented a true "you build it, you run it" approach. For most organizations, it is far from trivial to replicate true DevSecOps in practice since it is unlikely that they have access to same level of talent pool and resources as advanced organizations like Amazon, Google, Facebook, Apple, and Microsoft.

Platform Engineering Teams

The cognitive overload phenomenon in software engineering impacts development teams' ability to complete development tasks effectively because of the increased demand on them due to the tasks associated with DevSecOps and other infrastructure activities.

Cognitive Load theory (CLT) has been developed as an instructional design theory and studied the limitations of cognitive processing capacity and its effects on learning. When applied to software delivery, this concept concerns the performance of the development teams overloaded with many different tasks required by increasingly complex cloud native development, such as infrastructure provisioning, operating CI/CD pipelines, continuous testing, observability, security and developing and maintaining the services according to domain requirements.

Managing cognitive load on software delivery teams, particularly minimizing, or eliminating extraneous cognitive load, which is the load generated by the tasks not directly related to value of the product such as automating pipelines or developing Infrastructure as Code (IAC) scripts, became essential to increase productivity of the developers. With extraneous load minimized or eliminated, development teams can spend more time and energy to deal with core aspects of the product development such as understanding domain requirements or exploring libraries they need to utilize.

Organizations are increasingly choosing to create platform engineering teams to minimize extraneous load on development teams. The platform engineering teams are building the software development kits which are being utilized by the development team. This is not simply relabeling existing centralized infrastructure or support teams or ticket-driven operating models. These teams should be operating like the other product teams, be focused on the needs of their customers, in this case the development teams, and apply platform engineering best practices to deliver their services effectively, ideally as a self-service platform-as-a-product, referred as an "Internal Developer Platform (IDP)", which is reliable, fit-for-purpose and consumable easily by developers.

Platform Engineering and Team Interactions

In Team Topologies, the authors Skelton and Pais asserts that team responsibilities should match to the cognitive load that the team can handle and recommends a model which consists of four fundamental team types and three team interaction modes to manage cognitive load of the teams involved. According to Team Topologies model, stream-aligned team is the primary team type and these teams have end-to-end (design, develop and run) ownership of the product delivered to the customer. All other teams provide support to stream aligned teams to reduce their cognitive load and help them on core product delivery and support tasks:

- Enabling teams: Help stream aligned teams to improve capabilities by investing time on research, best practice development etc.
- Complicated-subsystem teams: Provide skills for a subsystem of the solution which requires specialist, hard-to-find knowledge such as specific algorithms, models etc.
- Platform team: Focus on delivery of all the capabilities required by stream aligned team to be able to deliver services or functionalities to their customers and as a result reducing their cognitive loads.

Internal Developer Platforms

Internal Developer Platforms built and maintained by platform engineering teams enable developer teams to deliver and operate solutions autonomously with reduced coordination. IDPs concentrate on creating an abstraction layer on top of the technologies and tooling software engineering teams must work with and automate recurring tasks they must perform. The end goal of an IDP is to reduce the cognitive pressure on the software delivery teams and increase the velocity of software delivery cycle.

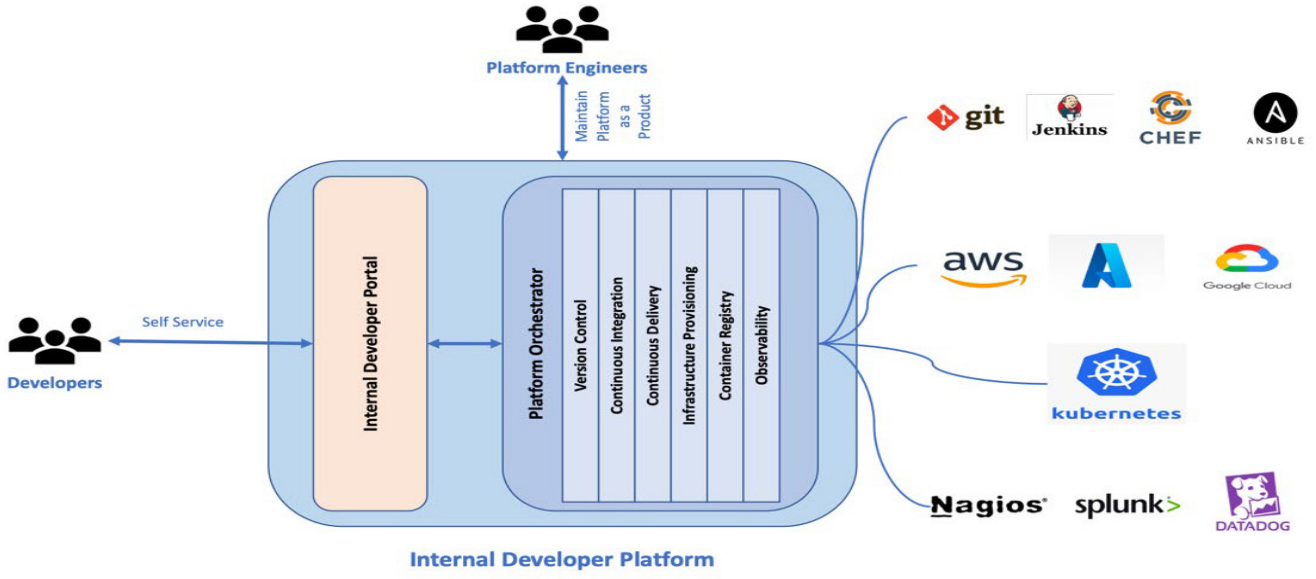


Figure 1Typical IDP Components

Typically, IDPs are built by integrating existing DevSecOps and infrastructure provisioning tools and provide capabilities which enable application teams to manage application configuration, provision infrastructure, create new environments, implement delivery pipelines, run the regression and security testing, and manage access based on developer roles. These capabilities are delivered via a Platform Orchestrator based on declarative application model or via Platform APIs. Mature IDPs also provide developer portals and service catalogs to ensure optimized developer experience.

IDP Capabilities

Capability	Functions performed
Developer Portal / Service Catalog	Primary interface to capabilities of the platform. It provides the unified view of everything developers need including documentation of best practices or so called "golden path" or "paved road".
Automated CICD	Automation of integration and delivery pipeline for Continuous Deployment (CD). IDPs enable developer to create and execute self-service workflows with all the steps they need such as GIT push, testing, code analysis and deployment to all environments. PE team can ensure these workflows conform to approved security and gating criteria via templates.
Automated CT	Integration of progression, regression, and security static code analysis testing as part of the build pipelines. PE teams can provide the test environment and integrate it with the test scripts before progressing the code to next SDLC life cycle.
Pre-integrated tools and services	Plug-ins for open source and cloud technologies across entire app stack and can be easily extended to integrated with new tools and products
Security	Security and compliance implementation integrated to service and infrastructure provisioning. CICD pipelines for all the apps, which include security scan stages to report the code for security vulnerabilities. Platform allows developer to select chosen security settings for the app e.g., auth mechanism, data privacy requirement, secure communication with other components along with other best practices.
Infrastructure and environment management	Enable developers to create cloud infrastructure by picking the whitelisted resources from different cloud providers and set up new environments without being IAC experts or waiting for Ops support.
Kubernetes Abstraction	Simplified K8 application deployment and management with higher level APIs such as Open Application Model (OAM - https://oam.dev/
Observability	Built-in observability for all the application developed and infrastructure components provisioned with monitoring, logging, and tracing
AI/ML insights	Integration with open-source AI tools to assist in several cognitive tasks e.g., elaborating the requirement specs from given business statement or generating documentation for a newly developed application
Operations Management	Enable developer teams to respond to incidents and provide Day-2 operations smoothly

Tools to build IDPs

IDPS can be built with open source or commercial tools which fall under different categories in software development ecosystem. Some of these tools are mature and already in use by many organizations in their CI/CD pipelines and the others are more recent in the market and positioning themselves specifically as IDP tools. Following list, from internaldevelopers.org, provides these categories and some of the tools within them:

- Integrated Development Environment (IDE) tools (Visual Studio Code, Eclipse, Xcode, etc.)
- Version Control Systems (VCS) (GitHub, GitLab, etc.)
- CI tools (Circle CI, GitHub Actions, Bitbucket)
- CT tools (SonarCube, Jmeter, etc)
- Container registries (Docker, Harbor, etc.)
- Platform orchestrators (Humanitec)
- Developer portals and service catalogs (Backstage, Port, Cortex)
- Kubernetes control planes (Ambassador Labs, Gimlet, Shipa)
- GitOps tools / CD controllers (ArgoCD, Kubevala, Jenkins, Flux CD)
- IaC (Terraform, Pulumi, etc.)
- Databases/storage
- DNS
- Managed or self-hosted Kubernetes
- Cloud providers
- Observability
- Security (Static Code Analysis, PEN Testing, Cryptography)

Platform Engineering Aspirational Maturity Model

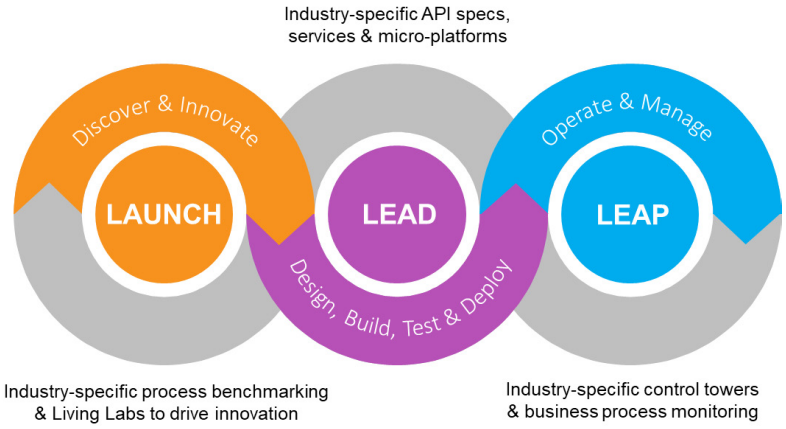
Platform Engineering platform maturity level can be measured based capabilities listed below (not limited to):

Capabilities	Level 1 (Beginner)	Level2 (Intermediate)	Level3 (Advanced)	Level 4 (Expert)
Developer Portal / Service Catalog	• Siloed self service capabilities • Self-service Provision • Documentation Ad-hoc, missing or out of date	• Partially Integrated self service capabilities • Partial set of software catalog and ecosystems to support across the organization needs • Self-service Automated Provision • Documentation standards established – platform allows creation of technical documentation	• Fully Integrated self service capabilities • Full set of software catalog and ecosystems to support across the organization needs • Self-service Automated Provision, Day2 Ops, Termination • Documentation as a code culture established – documentation is searchable	• Enable self-serve for anyone in the organization through all-in-one interface • Repeatable and responsive IDP
Automated CICD	• Automated < 50% • Frequency as required • Cycle Time > 2 hrs. • Impact on service < 2 hrs. • Post deployment testing - 30% automated • Deployment rollback - 90% manual, 10% auto • Manual request and approval	• Automated ~ 80% • Frequency on demand • Cycle Time < 2 hrs. • Impact on service < 1 hr. • Post deployment testing - 50% automated • Deployment rollback - 50% manual, 50% auto • Automated build request	• Automated ~ 100% • Frequency on demand • Cycle Time < 2 hrs. • No impact on user/ customer service • Post deployment testing - 100% automated • Deployment rollback - 100% auto • Canary or blue-green deployments • Approval based, gated workflow in automated pipeline	• Daily code promotion into pre-prod/UAT environment • Time based code promotion into production • Circuit breaker capabilities • Canary and blue green deployments • Fully automated request workflow
Automated CT	• Automated < 80% regression testing scripts • Automated < 50% progression testing scripts • Automated < 70 test data setup • Automated < 80 test environment setup	• Automated > 90% regression testing scripts • Automated < 50% progression testing scripts • Automated < 80 test data setup • Automated > 80 test environment setup. • Automated < 50% security testing	• Automated > 90% regression testing scripts • Automated < 80% progression testing scripts • Automated < 80 test data setup • Automated > 80 test environment setup • Automated < 80% security testing	• Automated > 90% regression testing scripts • Automated > 80% progression testing scripts • Automated < 80 test data setup • Automated > 90 test environment setup • Automated > 90% security testing
Pre-integrated Tools and services	• Manual or semi-automated tools and service • Standalone flows	• Partially integrated flows • Configurable integration templates	• Reusable, pre-configured, integrated services • Reusable enterprise application integration tools and platforms	• All-in-one iPaaS marketplace for internal and external users • Light weight enterprise services with dropdown points

Security	• IDE Security Plugins (e.g. SonarQube/ Fortify) • Manual SAST/DAST Testing	• Automated SAST/DAST • Security Technical Debt managed within SLA's	• Fully automated integrated security testing (XAST) and compliance • Dev consumable correlated vulnerability analysis, IoC/ TI STIX TAXII	• Auto Intrusion detection • RASP auto responds • Roll-back or toggle off • Block attacker /Shutdown services
Infrastructure and environment management	• Infrastructure provisioning, set up of environments and configuration management is in silos and mostly manual • Heavy dependency to infrastructure teams	• Multi-Cloud friendly • Decoupled apps from underlying infrastructure • IAC integrated • Environment standards established • Configuration files version managed	• Cloud resilient • Cloud agnostic • Developer can set up Environments themselves • Integrated infrastructure management, configuration management and environment set up	• Cloud native • Self-healing (app can detect, change /react in fully automated manner) • Integrated and fully automated infrastructure management, configuration management and environment set up
Kubernetes abstraction	• Polling builds • Manual tag & versioning • Some automated with virtualization	• Auto triggered builds • OS agnostic containerization	• Orchestrated and managed containerized apps (rollout/rollback) • Enables Zero downtime deployments (Including DB's) • Image repository with version management	• Serverless Build bakery • Zero touch continuous deployments in any environment and on any OS (Write once and run anywhere) • Run time security • Fully auto scalable
Observability	• Not real time • No traceability • Basic standard/limited /Manual alerts/ dashboard	• Visualization tool integration • Automated email notifications from CICD framework	• Real time metrics • Persona based notifications, alerts, and analytics dashboards	• Enterprise wise observability standards based on AI/ML • Anomaly detection, predictive analytics, auto healing • Palette based selection of multiple notification channels
AI/ML Insights	• Figure out the coding problems that cause bugs • Alerts PE teams so they can dig deeper • Manual / inconsistent and incomplete with ~ 50% incident detection	• Automated patterns to identify and recommendation to the developers • Automated alerts, notifications • Rule based ~75% incident detection (Tribal knowledge used for rules)	• Extensive normalization and enrichment • Assisted ML enriched and topology-based correlation • Consistent and ~ 95% incident detection	• Dynamic ML event processing • AI events/alerts • Very good incident detection coverage ~ 99%
Operations Management	• Ad-hoc processes for solving technical issues • Limited visibility into day-2-day ops	• Established process for incident response and some automation across the SDLC • Increased visibility into apps using tooling and processes (with ~75% alerts)	• Advanced anomaly detection capabilities and manage alerts (~95%) • Improved operational efficiency through cross team collaboration and learnings from production	• Continuous reliability with quality gates and managed alerts (~99%) • Automatically block progression of unreliable code • Streamlined feedback loops between production and non-production apps

Infosys Offerings for Platform Engineering

Infosys's approach is to drive innovation, business agility and hyper-productivity across innovate, transform, and operate phases of application development lifecycle which employs a collaborative, platform-based automation approach to achieve this.



The Infosys Live Enterprise Application Development Platform (LEAD) simplifies and accelerates the application modernization and development journey. It offers a range of features across five key modernization patterns – cloud native development, cloud modernization, database modernization, legacy modernization, and DevSecOps adoption. Features span the complete technology stack (UI, business layer, data, cloud services, infrastructure, etc.) and all application lifecycle stages including architecture, development, testing, site reliability engineering and deployment.

The platform provides guided workflows and AI-enabled tools to abstract the complexity of underlying technology, boosting developer productivity. It unlocks information from legacy systems, simplifies decision-making, and reduces dependency on niche skills, saving up to 40% effort and enabling up to 25% faster time-to-value. Deep insights into all types of technical debt enable high code quality from day-one. The platform also integrates with disparate ALM tools for data-driven insights that help improve sprint velocity, release predictability and product quality.

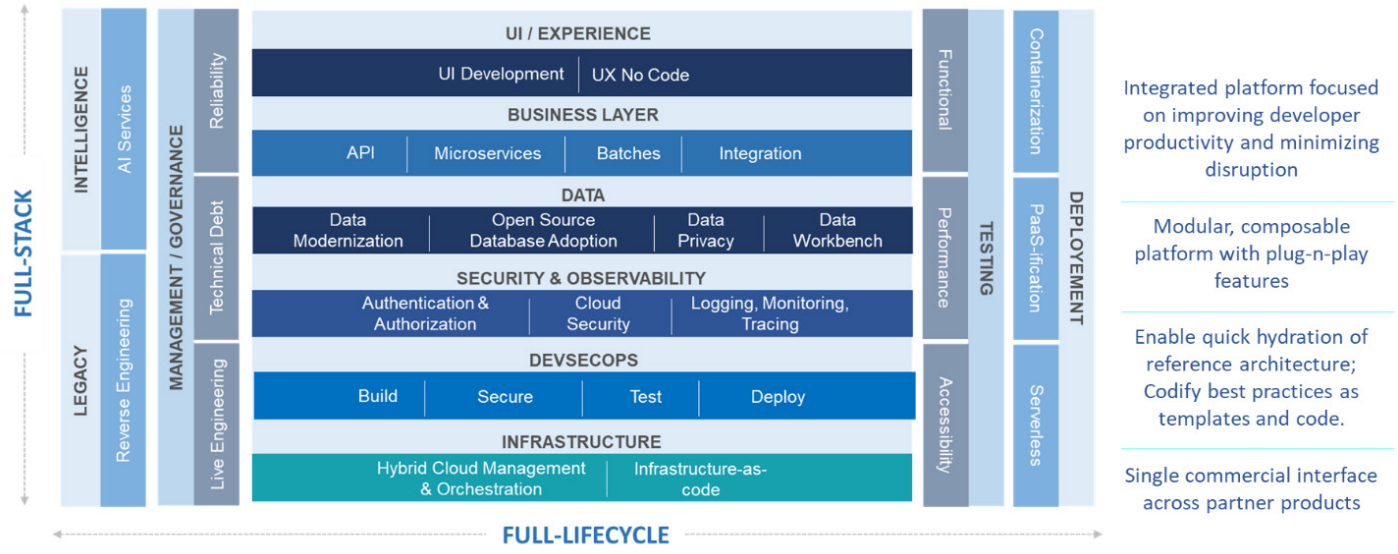


Figure 2 Lead Capability View

Infosys Leap provides a Cognitive First digital operations platform that makes business and IT operations integrated, agile, intelligent, predictive, efficient, and business aligned. Infosys Leap is powered by Cognitive Automation and Conversational AI, that makes operations zero touch and cost efficient. Infosys X also makes operations resilient through an AIOps solution with full stack observability supported by SRE (Site Reliability Engineering) principles that helps predict and proactively prevent issues before they impact business

Future Of Platform Engineering

As an emerging discipline Platform Engineering is already making significant impact on the way cloud native development is performed, particularly for DevOps and SRE related aspects. According to State of the DevOps Report, there is a strong correlation between usage of platform team model and DevOps evolution: 48% of the highly evolved teams (based on high performance on 4 key performance metrics from DORA - deployment frequency, lead time for changes, MTTR and change failure rate) use internal platforms as opposed to 25% and 8% of the teams at mid and low levels of their evolution respectively. The same resource also provides the following data points in relation to usage of self-service platforms: "Highly evolved firms make heavier use of internal platforms for their engineers, enabling developers to access authentication (62 percent), container orchestration (60 percent), and service-to-service authentication (53 percent), tracing and observability (49 percent), and logging request (47 percent) services via self-service."

We expect the popularity of platform teams and self-service developer platforms to grow driven by the following factors:

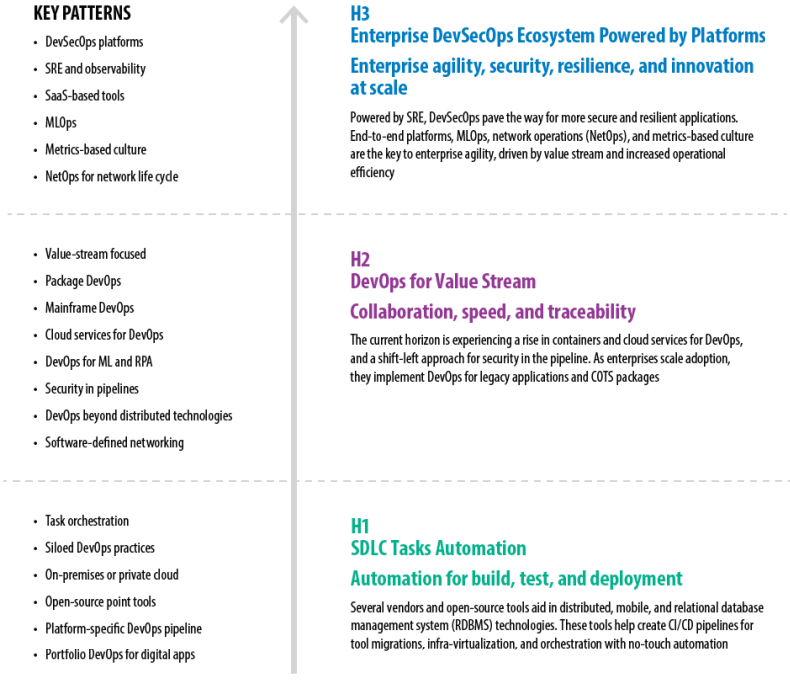
- Improving developer productivity and experience is becoming more critical for any organization developing software.
- Self-service developer platforms are increasingly becoming more attractive, especially for smaller organizations which do not have the resources with DevOps and SRE expertise.
- Kubernetes is already mainstream as the primary technology for cloud native development and its adoption will continue to grow. But the availability of expertise, internal or to hire from the market, is the main challenge. IDPs which provide tooling for Kubernetes abstraction are to play acritical role to deal with these challenges.
- Industry standards such as openTelemetry for observability, openAPI specification for describing and documenting APIs, and SPDX for software bill of materials continue to evolve. Platform teams and IDPs will make the adoption of these standards easier.
- Increased adoption of software and data engineering trends such as DataOps, MLOps, data mesh architecture relying on self-service data infrastructure, shift left security, privacy and testing, everything as a code (infrastructure, policy, configuration...) will force organizations to utilize PE best practices and IDPs.

DevOps, SRE and Platform Engineering

Platform Engineering is an outcome of DevOps evolution. Like DevOps and SRE, PE aims to reduce software delivery cycle time and increase reliability of the shipped products without making developer teams a bottleneck and allowing them to focus on developing the applications based on domain requirements.

As IKI's DevOps Compass indicates, at the Horizon 3 of the DevSecOps Evolution Continuum, enterprises *"adopt a framework with end-to-end platforms that provide cloud-agnostic DevSecOps automation as a service" with a "focus on scaling DevSecOps adoption through low code/no-code approaches, utilizing inbuilt and continuous security"*

Figure 1: Three horizons (H1, H2, and H3) in DevSecOps

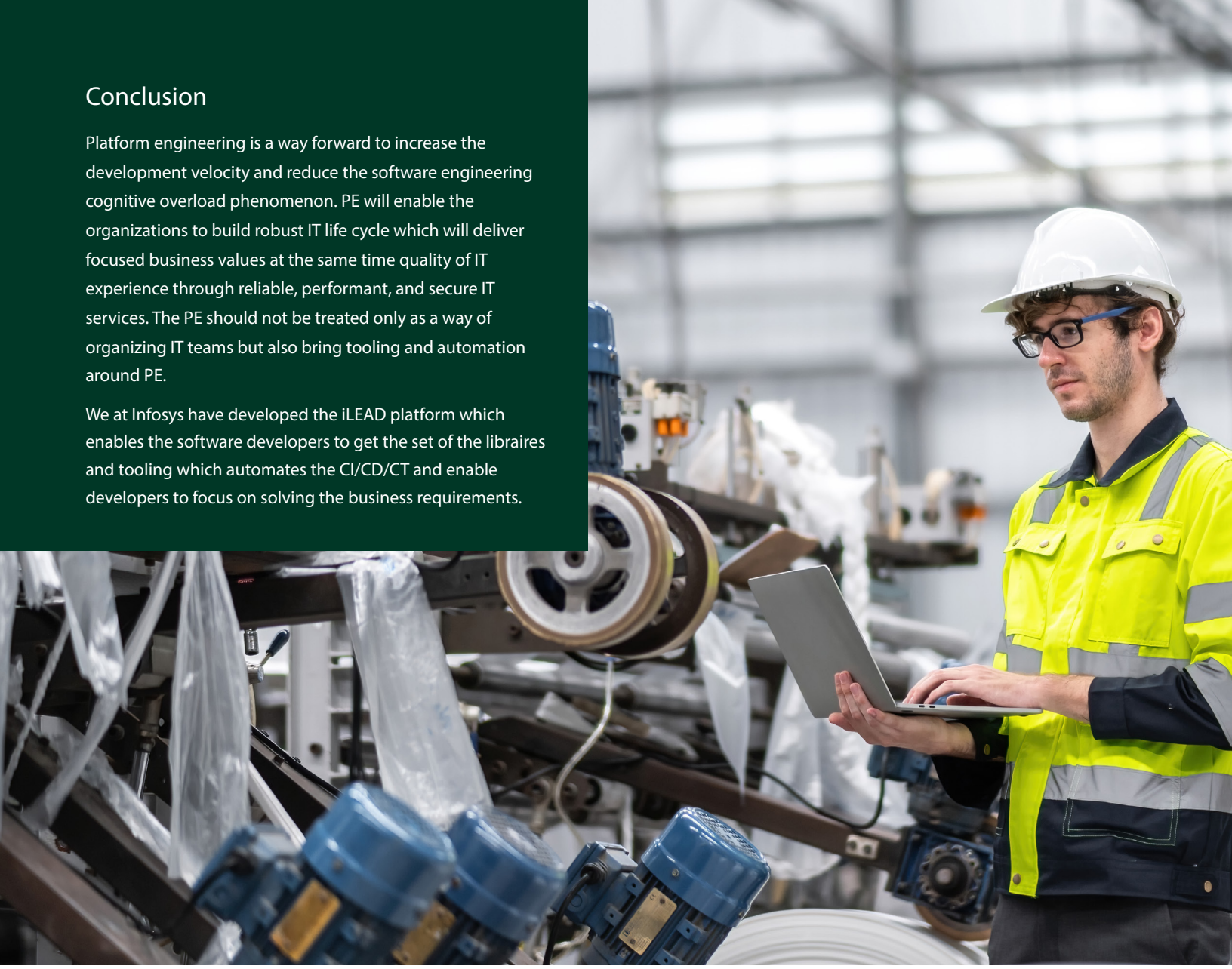


Source: Infosys

Conclusion

Platform engineering is a way forward to increase the development velocity and reduce the software engineering cognitive overload phenomenon. PE will enable the organizations to build robust IT life cycle which will deliver focused business values at the same time quality of IT experience through reliable, performant, and secure IT services. The PE should not be treated only as a way of organizing IT teams but also bring tooling and automation around PE.

We at Infosys have developed the iLEAD platform which enables the software developers to get the set of the libraires and tooling which automates the CI/CD/CT and enable developers to focus on solving the business requirements.



References

1 https://blog.matthewskelton.net/2013/10/22/what-team-structure-is-right-for-devops-to-flourish/

2 2021 State of DevOps Report presented by Puppet https://puppet.com/resources/report/2021-state-of-devops-report/

3 Cognitive Load During Problem Solving: Effects on Learning Cognitive Science. 12 (2): 257–285. CiteSeerX 10.1.1.459.9126

4 Team Topologies: organizing business and technology teams for fast flow - Matthew Skelton and Manuel Pais IT Revolution Press, Sept 2019

5 https://internaldeveloperplatform.org/

6 DEVSECOPS-Enterprise Value Catalyst for distributed Product Teams Infosys Knowledge Institute 2022

7 Technology Radar 2022 Thoughtworks 2022

8 https://about.gitlab.com/blog/2022/11/16/how-is-ai-ml-changing-devops/

9 https://www.bigpanda.io/magazine/build-a-boat-not-a-house-how-event-correlation-strengthens-your-observability-strategy/

10 https://psns.net/services/ObservabilityAssessment/

11 https://www.overops.com/blog/everything-you-need-to-know-about-the-4-stages-of-software-reliability/

About the Authors

Vishwanath Taware
VP - Unit Technology Officer

Ashim Bhuyan
Senior Principal - Enterprise Applications

Kiran Kumar Bathula
Senior Principal Technology Architect

Lalit Nayar
Senior Principal Technology Architect

Gokbora Uran
Senior Principal - Enterprise Applications

Prabhat Kumar
Senior Industry Principal

About the Contributors

Muthu Kumar Natarajan
Group Project Manager

Sriharsha Amalapurapu
Senior Enterprise Architect

For more information, contact askus@infosys.com

Infosys®
Navigate your next