Hype Cycle for Software Engineering, 2023

Published 1 August 2023 - ID G00791744 - 133 min read

By Analyst(s): Dave Micko, Joachim Herschmann, Mark O'Neill

Initiatives: Software Engineering Technologies; Adopt Modern Architectures and Technologies

As technologies such as AI-augmented software engineering and practices such as platform engineering take hold, software engineering is changing rapidly. Here, we explore software engineering practices that organizations are using to move from ideation to value creation.

Additional Perspectives

- ソフトウェア・エンジニアリングのハイプ・サイクル：2023年
(19 October 2023)
- Summary Translation: Hype Cycle for Software Engineering, 2023
(11 September2023)

More on This Topic
This is part of an in-depth collection of research. See the collection:

- 2023 Hype Cycles: Deglobalization, AI at the Cusp and Operational Sustainability

Strategic Planning Assumptions

- By 2026, 50% of software engineering leaders in enterprise companies will use product-centric delivery for all development initiatives, which is a significant increase from 15% today.
- By 2025, 30% of large enterprises will use cloud development environments to streamline development workflows and enable better manageability.
- By 2027, 50% of enterprise software engineers will use machine learning (ML)-powered coding tools, which is a major increase from fewer than 5% today.
- By 2026, 60% of the design of new websites and mobile apps will include content created by generative AI.
- By 2026, application security will become the most desired software engineering skill for more than 75% of organizations, which is a significant increase from fewer than 20% in 2022.

Analysis

What You Need to Know

This document was revised on 22 August 2023. The document you are viewing is the corrected version. For more information, see the Corrections *page on gartner.com.*

Software engineers and leaders can use the new 2023 Hype Cycle for Software Engineering to understand upcoming trends, and to appreciate the ways innovations are affecting the entire software engineering ecosystem. This Hype Cycle covers the things that are just being invented, the things that are being overhyped, the innovations that are disappointing developers and those that are providing long-term utility.

Software engineering leaders should:

- Explore AI-augmented software engineering, and learn how these technologies can open up new opportunities for value creation, as well as create new challenges for the software engineering organization.
- Develop plans to assess how value stream management platforms, internal developer portals, software bills of materials (BOMs) and open-source program offices can add value to their organizations.
- Leverage the power of their organizations' business technologists now, by turning them into citizen developers with low-code application platforms.

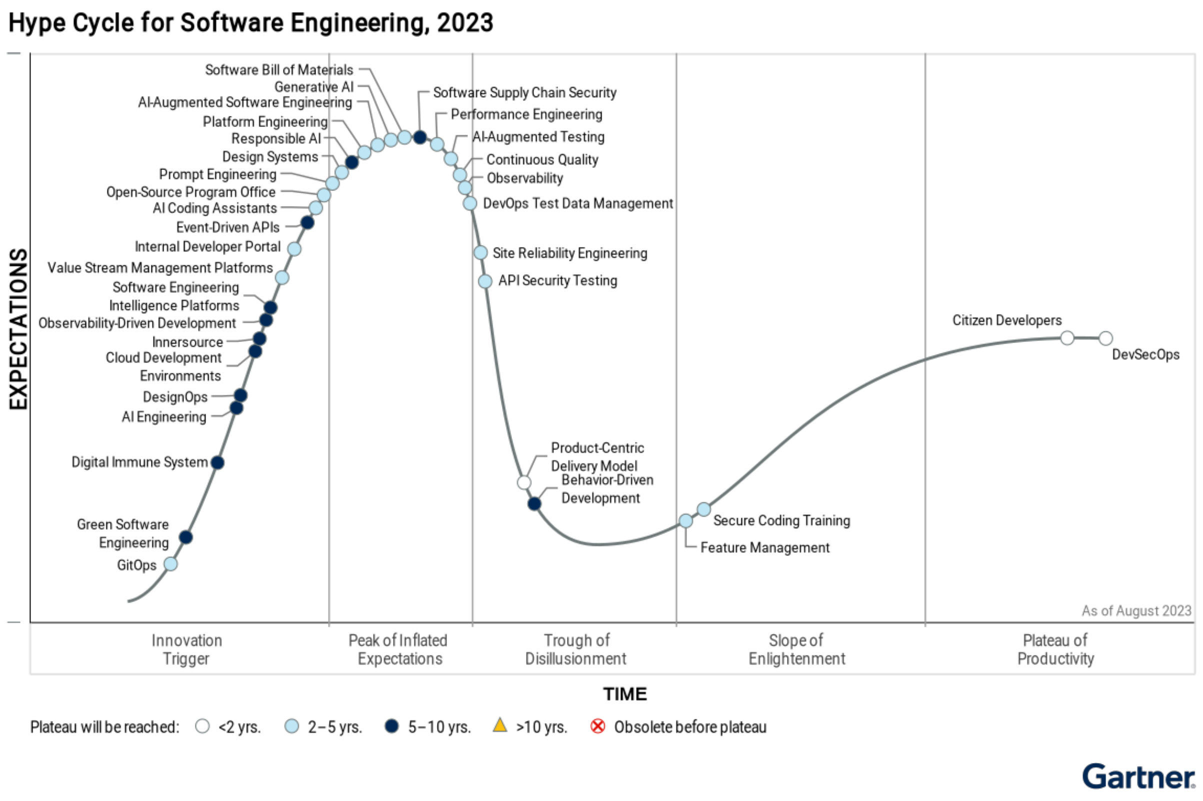
The Hype Cycle

Gartner's Hype Cycle for Software Engineering tracks key approaches and technologies that help software engineering leaders lead teams to deliver business value faster, fill gaps in their practices and support their engineers with better tools. In this year's Hype Cycle, many of the entries have made significant progress toward utility and adoption, reflecting a rapid and evolving ecosystem of new technologies and practices:

- **AI, ML and large language models:** Organizations should immediately begin to set policies, guardrails and golden paths, giving guidance to software engineers in the approved use of AI-augmented software engineering, AI-augmented software development, AI coding assistants, testing and prompt engineering.

- **Platforms, pipelines and observability:** Continuing the trend toward focusing on optimizing the developer's experience and winning the war for engineering talent, teams should use inner-source techniques to apply open-source practices to internal development. Teams should also consider deploying robust observability practices and platforms, ultimately allowing for observability-driven development.
- **Delivering value:** Organizations are implementing software engineering intelligence platforms that ingest data from the entire software engineering and delivery process to visualize and optimize the flow of work and optimize throughput.
- **Security:** Organizations are recognizing the importance of securing open-source software (OSS) and creating OSS program offices to govern their use of OSS at scale. Software supply chain security, software BOMs and API security are now critical components of DevSecOps.
- **Quality:** Leaders should think of quality in terms of business outcomes and customer value, rather than outputs and simple testing. Creating and managing test data are increasingly important.
- **Green software engineering:** As organizations produce and consume more software as part of digital initiatives, the increase in carbon footprint from compute-intensive workloads can be at odds with their sustainability targets. This Hype Cycle introduces green software engineering to shift the mindset from treating the environmental impact of software as an afterthought to incorporating green practices right from the beginning.
- **Resilience and Performance:** As customers have come to expect performance and reliability in their digital experiences, the need for site reliability engineering (SRE) has grown. Digital immune system interlinks practices from six areas: observability, software testing, chaos engineering, software development, SRE and software supply chain security. This ensures high resiliency and quality of applications.

Figure 1: Hype Cycle for Software Engineering, 2023



The Priority Matrix

The Priority Matrix shows the benefit levels and the number of years to mainstream adoption for the technologies presented in this Hype Cycle.

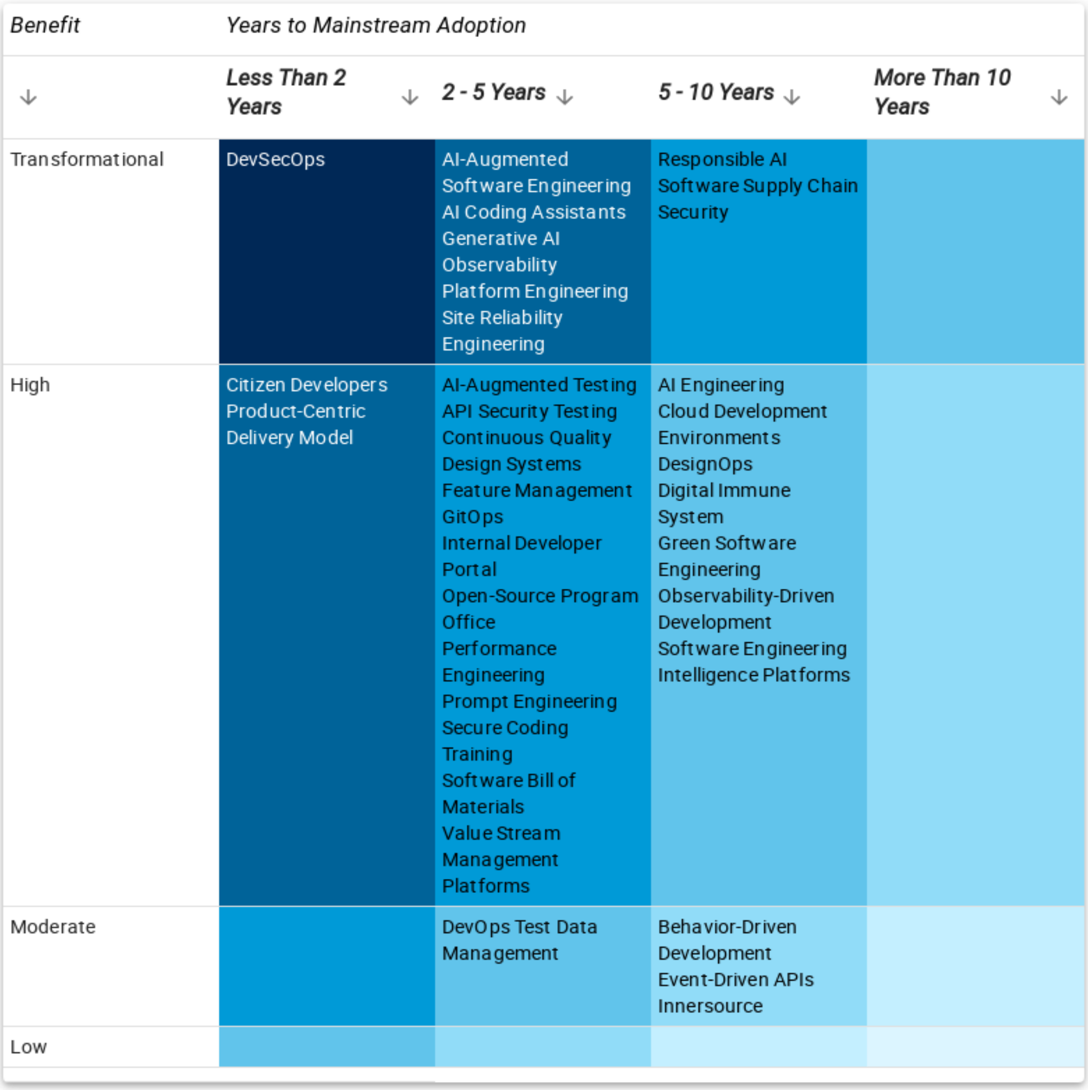
Transformational innovations can have a significant impact on an organization's business models, driving new strategies and tactics. AI-augmented and ML-powered software engineering is changing the way software is being created, tested and operated, and the need for responsible AI is growing. Practices such as observability, platform engineering and SRE will begin injecting insights from deployed systems into the systems being developed. Leaders need to document and secure their entire software supply chain, as OSS is increasingly being leveraged in the enterprise.

High-benefit innovations are less likely to change an organization's business model, but will have a significant impact on software engineering practices. Pay particular attention to the innovations with the highest benefits, such as internal developer portals, AI-augmented testing, and the increasing use and utility of design systems.

Moderate-benefit innovations have a narrower scope, but a deep impact. Leaders should work with their functional leads to identify which of the innovations in this category they should apply. In this category, they should prioritize enterprise innersourcing practices, DevOps test data management and event-driven APIs.

Table 1: Priority Matrix for Software Engineering, 2023

(Enlarged table in Appendix)



Benefit	*Years to Mainstream Adoption*			
	Less Than 2 Years	*2 - 5 Years*	*5 - 10 Years*	*More Than 10 Years*
Transformational	DevSecOps	AI-Augmented Software Engineering AI Coding Assistants Generative AI Observability Platform Engineering Site Reliability Engineering	Responsible AI Software Supply Chain Security	
High	Citizen Developers Product-Centric Delivery Model	AI-Augmented Testing API Security Testing Continuous Quality Design Systems Feature Management GitOps Internal Developer Portal Open-Source Program Office Performance Engineering Prompt Engineering Secure Coding Training Software Bill of Materials Value Stream Management Platforms	AI Engineering Cloud Development Environments DesignOps Digital Immune System Green Software Engineering Observability-Driven Development Software Engineering Intelligence Platforms	
Moderate		DevOps Test Data Management	Behavior-Driven Development Event-Driven APIs Innersource	
Low				

Source: Gartner (August 2023)

Off the Hype Cycle

On the Rise

GitOps

Analysis By: Paul Delory, Arun Chandrasekaran

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Early mainstream

Definition:

GitOps is a type of closed-loop control system for cloud-native applications. The term is often used more expansively, usually as a shorthand for automated operations or CI/CD, but this is incorrect. According to the canonical OpenGitOps standard, the state of any system managed by GitOps must be: (1) expressed declaratively, (2) versioned and immutable, (3) pulled automatically, and (4) continuously reconciled. These ideas are not new, but new tools and practices now bring GitOps within reach.

Why This Is Important

GitOps can be transformative. GitOps workflows deploy a verified and traceable configuration (such as a container definition) into a runtime environment, bringing code to production with only a Git pull request. All changes flow through Git, where they are version-controlled, immutable and auditable. Developers interact only with Git, using abstract, declarative logic. GitOps extends a common control plane across Kubernetes (K8s) clusters, which is increasingly important as clusters proliferate.

Business Impact

By operationalizing infrastructure as code, GitOps enhances availability and resilience of services:

- GitOps can be used to improve version control, automation, consistency, collaboration and compliance.
- Artifacts are reusable and can be modularized.
- Configuration of clusters or systems can be updated dynamically. All of this translates to business agility and a faster time to market.

- GitOps artifacts are version-controlled and stored in a central repository, making them easy to verify and audit.

Drivers

- **Kubernetes adoption and maturity:** GitOps must be underpinned by an ecosystem of technologies, including tools for automation, infrastructure as code, continuous integration/continuous deployment (CI/CD), observability and compliance. Kubernetes has emerged as a common substrate for cloud-native applications. This provides a ready-made foundation for GitOps. As Kubernetes adoption grows within the enterprise, so can GitOps, too.

- **Need for increased speed and agility:** Speed and agility of software delivery are critical metrics that CIOs care about. As a result, IT organizations are pursuing better collaboration between infrastructure and operations (I&O) and development teams to drive shorter development cycles, faster delivery and increased deployment frequency. This will enable organizations to respond immediately to market changes, handle workload failures better, and tap into new market opportunities. GitOps is the latest way to drive this type of cross-team collaboration.

- **Need for increased reliability:** Speed without reliability is useless. The key to increased software quality is effective governance, accountability, collaboration and automation. GitOps can enable this through transparent processes and common workflows across development and I&O teams. Automated change management helps to avoid costly human errors that can result in poor software quality and downtime.

- **Talent retention:** Organizations adopting GitOps have an opportunity to upskill existing staff for more automation- and code-oriented I&O roles. This opens up opportunities for staff to learn new skills and technologies, resulting in higher employee satisfaction and retention.

- **Cultural change:** By breaking down organizational silos, development and operations leaders can build cross-functional knowledge and collaboration skills across their teams to enable them to work effectively across boundaries.

- **Cost reduction:** Automation of infrastructure eliminates manual tasks and rework, improving productivity and reducing downtimes, both of which can contribute to cost reduction.

Obstacles

- **Prerequisites:** GitOps is only for cloud-native applications. Many GitOps tools and techniques assume the system is built on Kubernetes (frequently, they also assume that a host of other technologies are built on top of K8s). By definition, GitOps requires software agents to act as listeners for changes and help to implement them. GitOps is possible outside Kubernetes, but in practice K8s will almost certainly be used. Thus, GitOps is necessarily limited in scope.
- **Cultural change:** GitOps requires a cultural change that organizations need to invest in. IT leaders need to embrace process change. This requires discipline and commitment from all participants to doing things in a new way.
- **Skills gaps:** GitOps requires automation and software development skills, which many I&O teams lack.
- **Organizational inertia:** GitOps requires collaboration among different teams. This requires trust among these teams for GitOps to be successful.

User Recommendations

- **Target cloud-native workloads initially:** Your first use case for GitOps should be operating a containerized, cloud-native application that is already using both Kubernetes and a continuous delivery platform such as Flux or ArgoCD.
- **Build an internal operating platform:** This is the foundation of your GitOps efforts. Your platform should manage the underlying infrastructure and deployment pipelines, while enforcing security and policy compliance.
- **Embed security into GitOps workflows:** Security teams need to shift left, so the organization can build holistic CI/CD pipelines that deliver software and configure infrastructure, with security embedded in every layer.
- **Be wary of vendors trying to sell you GitOps:** GitOps isn't a product you can buy, but a workflow and a mindset shift that becomes part of your overall DevOps culture. Tools that expressly enable GitOps can be helpful; but GitOps can be done with nothing more than standard continuous delivery tools that support Git-based automation.

Sample Vendors

GitLab; Harness; Red Hat; Upbound; Weaveworks

Gartner Recommended Reading

Innovation Insight: Top 4 Use Cases for GitOps

Is Using GitOps-Style Automation With Kubernetes Right for Me?

How to Scale DevOps Workflows in Multicluster Kubernetes Environments

Designing and Operating DevOps Workflows to Deploy Containerized Applications With Kubernetes

Automate the Application Delivery Value Stream

Green Software Engineering

Analysis By: Manjunath Bhat, Mukul Saha

Benefit Rating: High

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

Green software engineering is the discipline of building software that is carbon-efficient and carbon-aware. Carbon efficiency is about producing the least carbon possible over the software development life cycle. Carbon awareness is about optimizing software to use low-carbon sources of energy. Building green software involves making energy-efficient choices of architecture and design patterns, algorithms, data structures, programming languages, language runtimes and infrastructure.

Why This Is Important

Green software engineering shifts the mindset from treating the environmental impact of software as an afterthought to incorporating green practices from the get-go. As organizations produce and consume more software as part of digital initiatives, the increase in carbon footprint from compute-intensive workloads can be at odds with their sustainability targets. Green software engineering practices are a subset of the overall discipline of building sustainable software.

Business Impact

Green software engineering can help organizations meet their strategic sustainability goals and improve their brand reputation, ensure regulatory compliance and drive resource efficiencies. Green principles can also shape product roadmaps — product owners may choose to deprioritize features that negatively impact the organization's sustainability footprint. The positive side effects may also include improved performance and uncluttered UX.

Drivers

- **Open-source communities and contributions:** Green software engineering has an increased impetus from open-source communities such as the Green Software Foundation (GSF), a nonprofit formed under the Linux Foundation. The GSF is building an ecosystem of people, standards and tooling and establishing patterns, practices and principles for building green software. Examples of GSF-backed projects include carbon-aware SDK, green software patterns, green software practitioner training and software carbon intensity specification.

- **Contractual obligations:** Government agencies in some countries as well as private companies are including clauses in their IT outsourcing and procurement contracts that require providers to demonstrate commitment to sustainability goals. When organizations set targets to reduce carbon emissions, it has a ripple effect on their supply chain. This incentivizes software development organizations to adopt green software engineering practices.

- **Sustainability initiatives:** The 2022 Gartner Sustainability Opportunities, Risks and Technologies Survey shows that sustainability investment is likely increasing despite disruption and tightening economic conditions. Most (87%) respondents indicated they anticipate increasing their organization's investment in sustainability over the next two years. The same survey indicates that improved company reputation, improved resource efficiency and enhanced customer satisfaction and engagement are the top three benefits organizations have achieved to date from their sustainability programs.

- **Cloud carbon footprint:** The improved visibility into cloud carbon footprint helps organizations to measure the environmental impact of cloud workloads. Public cloud providers and third-party tools (e.g., Thoughtworks' Cloud Carbon Footprint) provide carbon footprint dashboards to measure carbon emissions for specific workloads. Generative AI-based cloud services will further increase the significance of green software engineering.

Obstacles

- **Conflicting priorities:** Building green software is a business decision, balancing the needs of stakeholders such as customers, investors, staff, regulators or partners. Many software sustainability investments imply business trade-offs (e.g., making a website more energy-efficient might mean not using video, impacting online sales) See Is Sustainable Software a Distraction or an Imperative?.

- **Conflicting capabilities:** Programming language choice is often about support, compatibility, interoperability and a team's proficiency in the language, not sustainability. Some languages and frameworks that are inefficient in energy consumption are also popular and productive, such as Python or Ruby. Languages such as Rust and C/C++ may be energy-efficient but may not be suited to the task. Also, making energy-efficient architecture choices is harder because it may involve dependency on underlying hardware and deployment environments (e.g., cloud, data centers, devices, edge).

- **Culture change:** The lack of incentives, funding and an executive "sponsor" could deter software engineering teams from aggressively pursuing green practices.

User Recommendations

- Develop a green software strategy with business stakeholders by identifying where software sustainability matters to them, what trade-offs they'll accept to achieve it and what success metrics they require.

- Form communities of practice to disseminate knowledge and working examples; share lessons learned; and build organizational competence to start and scale green software engineering initiatives.

- Encourage platform teams to create a curated catalog of patterns, practices, templates and tooling to scale cross-cutting green software engineering capabilities across the organization. Adopt a Team Topologies approach and create enabling teams to advocate for carbon reductions, demonstrate proof of value and make it easier to integrate green practices for product teams in specific contexts.

Sample Vendors

CAST; Google; Microsoft; Sonar; Thoughtworks

Gartner Recommended Reading

Is Sustainable Software a Distraction or an Imperative?

Sustainability: A Customer Priority and Provider Imperative

Quick Answer: How Green Are Public Cloud Providers?

Predicts 2023: Sustainability — It's Complicated

2023 Growth Agenda: Sustainability Is an Opportunity for Business Growth

Digital Immune System

Analysis By: Joachim Herschmann

Benefit Rating: High

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

The digital immune system (DIS) approach interlinks practices from six areas: observability, software testing, chaos engineering, automated remediation, site reliability engineering (SRE) and software supply chain security. This ensures high resiliency and quality of applications. It focuses on making applications more resilient so that they recover quickly from failures, rather than impacting users and business performance.

Why This Is Important

Software development teams are under pressure to deliver faster, while dealing with increasingly complex architectures, compliance needs and constantly evolving technology stacks. This exposes organizations to operational and business risks, when applications and services are severely compromised or stop working altogether. The DIS approach combines and amplifies several resilience engineering practices, to mitigate these risks and deliver a high level of business impact.

Business Impact

A digital immune system strategy helps engineering teams build and deploy software confidently, as they can continuously assess system health, remediate issues and optimize system performance. Also, systems designed for digital immunity help identify code quality issues related to performance, security and other operational attributes early in the software development life cycle. This creates software engineering teams that are focused on connecting software development to business value.

Drivers

- **Highly distributed, cloud-based applications:** Software engineering leaders struggle to plan for all eventualities of how modern, highly distributed software systems may fail. Troubleshooting issues in distributed applications requires fundamentally different techniques, compared with monolithic or client/server applications.

- **Business continuity:** As an enabler of business operations, IT plays a significant role in business continuity and enterprise success. In response to pressure from business stakeholders, software engineering leaders are looking for new ways to improve the resilience of business-critical systems and reduce their vulnerability.

- **Risk mitigation:** Prominent examples of how leading digital enterprises have suffered from software glitches, outages or security incidents put additional pressure on software engineering leaders to mitigate risks to the business.

- **Continuous improvement:** Organizations need a frame of reference to improve the resilience of digital systems, measure progress and use failures as learning opportunities. Teams must continuously improve their skills to minimize the impact of a failure, with a focus on preserving the overall health of the system.

- **Developer experience:** Developers often get disrupted in their daily work, as they have to deal with incidents and frequent firefighting exercises, which are detrimental to developer experience. A DIS approach fosters a mindset of continuous quality improvement, and enables a shift toward building quality and security into the product.

- **Reliability as a key differentiator:** Organizations that have recognized reliability as a differentiator in the market need key insights, operational agility and competence. They acquire these through the combination of observability, AI-augmented testing, chaos engineering, SRE and security practices.

Obstacles

- **Technology focus:** Tools are enablers of a digital immune system strategy, but just throwing technology at a problem will not solve it. Tools automate and facilitate your processes and practices, but they need to be in place first and evolve over time.

- **Failure to measure:** The core elements of a digital immune system are not new. But most organizations are only using some parts of it, often in uncoordinated ways, and without a plan to close gaps and evolve them into a more holistic strategy. It is, therefore, important to measure progress in the evolution of the different practices and the success of technologies that enable them.

- **Internal pushback:** Implementing a digital immune system requires engaging stakeholders across the organization and seeking out opportunities for collaboration and improvement. Such a holistic approach can be seen as too far-reaching and lead to turf wars.

User Recommendations

- Adopt a DIS strategy as a standard software engineering approach to prepare their teams to handle unexpected and unforeseeable system behaviors, quickly remediate software defects, and minimize the impact on users. Support a "you build it, you run it" (YBIYRI) approach.

- Assess which business capabilities have the highest priority or will benefit the most from DIS investments.

- Establish a platform engineering team to build standard platform support for a DIS. Create dedicated communities of practice (CoPs) to share lessons learned, guiding principles, reusable assets, standards, tools and any AI-based insights realized.

- Encourage and reward resilience improvements across the organization, especially collaboration on DIS opportunities.

- Foster a collaborative culture between development, security and operations teams to ensure ongoing support for these initiatives.

Gartner Recommended Reading

Top Strategic Technology Trends for 2023: Digital Immune System

Improve Software Quality by Building Digital Immunity

AI Engineering

Analysis By: Kevin Gabbard, Soyeb Barot

Benefit Rating: High

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

AI engineering is foundational for enterprise delivery of AI solutions at scale. The discipline unifies DataOps, MLOps and DevOps pipelines to create coherent enterprise development, delivery (hybrid, multicloud, edge), and operational (streaming, batch) AI-based systems.

Why This Is Important

The potential value of AI has led to huge demand to rapidly launch market-ready AI solutions. This is a big engineering challenge. Most enterprises still struggle to move individual pilots to production, much less operate portfolios of AI solutions at scale. Establishing consistent AI pipelines enables enterprises to develop, deploy, adapt and maintain AI models (statistical, machine learning, generative, deep learning, graph, linguistic and rule-based) consistently, regardless of environment.

Business Impact

AI engineering enables organizations to establish and grow high-value portfolios of AI solutions. Most AI development is currently limited by significant operational bottlenecks. With AI engineering methods — DataOps, ModelOps and DevOps — it is possible to deploy models into production in a structured, repeatable factory-model framework to realize significant value.

Drivers

- Intense hype surrounding generative AI solutions is increasing the overall demand for AI-powered solutions.
- DataOps, ModelOps and DevOps provide best practices for moving artifacts through the AI development life cycle. Standardization across data and model pipelines accelerates the delivery of AI solutions.
- The elimination of traditional siloed approaches to data management and AI engineering reduces impedance mismatch across data ingestion, processing, model engineering and deployment, which inevitably drift once the AI models are in production.
- AI engineering enables discoverable, composable and reusable AI artifacts (data catalogs, code repositories, reference architectures, feature stores, model stores, etc.) across the enterprise context. These are essential for scaling AI enterprisewide.
- AI engineering makes it possible to orchestrate solutions across hybrid, multicloud, edge AI or IoT.
- Broader use of foundational platforms provides initial success at scaling the production of AI initiatives with existing data, analytics and governance frameworks.

Obstacles

- Sponsorship for foundational enterprisewide AI initiatives is unclear.
- AI engineering requires simultaneous development of pipelines across domains.
- AI engineering requires integrating full-featured solutions with select tools, including open-source technologies, to address enterprise architecture gaps with minimal functional overlap. These include gaps around ETL stores, feature stores, model stores, model monitoring, pipeline observability and governance.
- AI engineering requires cloud maturity and possible rearchitecting, or the ability to integrate data and AI model pipelines across deployment contexts. Potential complexity and management of analytical and AI workloads alongside costs may deter organizations that are in the initial phases of AI initiatives.
- Enterprises often seek “unicorn” experts to productize AI platforms. Few vendors provide AI engineering capabilities, making such skills hard to find. Enterprises often have to build and support these environments on their own.

User Recommendations

- Establish a leadership mandate for enterprisewide foundational AI initiatives.
- Maximize business value from ongoing AI initiatives by establishing AI engineering practices that streamline the data, model and implementation pipelines.
- Simplify data and analytics pipelines by identifying the capabilities required to operationalize end-to-end AI platforms and build AI-specific toolchains.
- Use point solutions sparingly and only to plug feature/capability gaps in fully featured DataOps, MLOps, ModelOps and PlatformOps tools.
- Develop AI model management and governance practices that align model performance, human behavior and delivery of business value. Make it easier for users to adopt AI models.
- Leverage cloud service provider environments as foundational to build AI engineering. At the same time, rationalize your data, analytics and AI portfolios as you migrate to the cloud.
- Upskill data engineering and platform engineering teams to adopt tools and processes that drive continuous integration/continuous development for AI artifacts.

Sample Vendors

Amazon Web Services; Dataiku; DataRobot; Domino Data Lab; Google; HPE; IBM; Iguazio; Microsoft

Gartner Recommended Reading

Top Strategic Technology Trends for 2022: AI Engineering

Demystifying XOps: DataOps, MLOps, ModelOps, AIOps and Platform Ops for AI

A CTO's Guide to Top Artificial Intelligence Engineering Practices

Cool Vendors in Enterprise AI Operationalization and Engineering

Cool Vendors in AI Core Technologies — Scaling AI in the Enterprise

DesignOps

Analysis By: Will Grant, Brent Stewart

Benefit Rating: High

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

DesignOps is a set of operational practices that enables design-team management and product-level delivery of design assets. The team management side stresses strategic alignment with business operations for the central design function and career development. The product delivery side combines user experience (UX), product management and technology operations to enable efficient and DevOps-compatible plans, estimates and processes that support quality, collaboration and ongoing innovation.

Why This Is Important

DesignOps introduces formalized approaches to governance, operations and people management. As a set of easy-to-use operational standards, DesignOps continues to gain in popularity. Digital product companies and agencies are discovering the tremendous value of a proven operational approach for UX team management and design delivery on product teams.

Business Impact

DesignOps represents the first widespread implementation of operational methods and techniques created for both designers and developers. DesignOps adds value during the creation and delivery of design assets. DesignOps practices should cover how teams are organized such that they can better support ongoing feature enhancement and idea generation without interrupting the continuous workflow of development teams.

Drivers

- **Innovation:** When coupled with DevOps, DesignOps leads to more innovative solutions. As a practice, DesignOps employs dual-track agile, which sets aside ongoing tracks of work dedicated to new discovery, idea generation and design exploration. This work acts as a constant source of evidence-based, multidisciplinary innovation.

- **Speed:** DesignOps reduces the time to market for major updates and incremental feature enhancements alike. Due to the concepts of continuous discovery and continuous delivery, developers engage in tech design, architectural explorations and proofs of concept (POCs) sooner than before, and with a deeper understanding of the overall vision.

- **Collaboration:** DesignOps increases communication and camaraderie between design and development teams. The design-development gap exists for many reasons, one of them being culture. DesignOps promotes multidisciplinary teams in workshop settings, design sprints or one-on-one “pairing and sharing” that promotes understanding, empathy and relationship building between these two crucially important groups.

Obstacles

- Few UX practitioners are educated in detailed planning and estimation, using a common work breakdown structure (WBS).

- Few product managers are trained in UX planning, estimating and tracking, and many of the design platforms lack robust change control solutions, although this is improving.

- Popular enterprise agile planning (EAP) tools are not designed with UX practitioners, activities and deliverables in mind (although this is changing), leading to resistance from UX teams to adopt tools they perceive as “for developers.”

User Recommendations

Software engineering leaders should:

- Encourage design and UX professionals to adopt a DesignOps practice to better manage the complete design life cycle.

- Ensure that the DesignOps practice covers the following three key aspects: how UX teams are organized, the tools and processes for delivering UX artifacts and how success is measured.
- Develop Agile training designed for UX professionals, available as part of the DesignOps practice.
- Determine the value of a DesignOps approach with a pilot program involving an existing high-performing team.

Following a successful pilot, application leaders and the pilot team members should:

- Engage in a productwide rollout that involves training, updated product plans and the allocation of one or more people to the role of design manager.

It should be noted that a successful rollout of DesignOps at the product level requires buy-in from product management, design and development teams, as well as robust logistical and administrative skills.

Gartner Recommended Reading

How Design, Development and Product Management Can Work Together Successfully

The 4 Secrets of Design-Led Companies

2023 Software Engineering Primer: Build and Deliver New Digital Products/Experiences to Drive Business Results

Keys to DevOps Success

Cloud Development Environments

Analysis By: Manjunath Bhat

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Emerging

Definition:

Cloud development environments (CDEs) provide remote, ready-to-use access to a cloud-hosted development environment with minimal effort for setup and configuration. This decoupling of the development workspace from the physical workstation enables a low-friction, consistent developer experience. CDEs comprise elements of a traditional IDE, such as code editing, debugging, code review and code collaboration. They increasingly include ML-based coding assistants and integrate with DevOps platforms.

Why This Is Important

CDEs provide consistent, secure developer access to preconfigured development workspaces. This frees them from setting up their own local environments, eliminating the need to install and maintain dependencies, software development kits, security patches and plug-ins. CDEs are prepackaged with language tooling for multiple programming languages, enabling teams to write code for different application stacks with standardized and templatized workflows.

Business Impact

CDEs are becoming popular due to their seamless integration with Git-based repositories and continuous integration/continuous delivery (CI/CD) tools, thus enhancing developer productivity and developer experience. They also enable security and platform teams to govern and secure user access to development environments, even from personally-owned devices. This mitigates the security and operational risks. CDEs provide a scalable way to support a distributed workforce and hybrid work environments.

Drivers

Gartner predicts that by 2026, 60% of cloud workloads will be built and deployed using CDEs. Four factors are driving their increased adoption:

- Remote work and remote onboarding of software developers create a need for a frictionless onboarding experience. The ability to share the development environment among distributed team members makes remote debugging and pair programming easier.
- Organizations increasingly need custom configured hardware to support specific application architectures, such as Apple M1/M2 silicon for mobile app development or GPU support for data and analytics workloads.

- The ability to centrally manage, govern and secure development environments has become especially important to minimize the threat of software supply chain attacks. In addition, CDEs make it easier to support and secure bring-your-own-device use cases.
- Automating DevOps workflows introduces more plug-ins, extensions and API integrations, which makes the distribution of updates and configurations to local machines cumbersome and unreliable.
- Building modern, distributed applications such as mesh services, event-driven apps and Kubernetes applications introduce complexities, making it harder to develop and test on local machines.

Obstacles

- CDEs incur costs in addition to what an organization may already be paying for DevOps tooling. The cost can be prohibitive for teams that rely on open-source development tools for application development and delivery on local machines.
- Connectivity presents another obstacle. Poor or inconsistent internet speeds adversely affect developer experience. In many cases, developers simply like to use their own local machines to get their work done. That allows them to use their own plugins, editors and scripts, all of which will be difficult through a browser-based interface.
- Security and compliance policies can prevent the use of cloud for development in some organizations. This can rule out CDEs that rely on public cloud services. Note that CDEs don't necessarily have to be provisioned in public cloud environments.
- There may be resistance from developers since it may impede their ability to research, experiment and innovate using full or elevated permissions on local development machines.

User Recommendations

Software engineering leaders should:

- Pilot the use of CDEs when their teams are developing with cloud-hosted source repositories. Plan for downtime resulting from service outages, since CDEs rely on the remote development environment being up and running.

- Ensure that CDEs are part of an overall developer self-service platform with centralized governance. The benefit of centralized governance and developer agility can be a win-win for platform and product teams.
- Use CDEs as one of the “quick wins” to improve the onboarding experience for new developers and reduce ramp-up time.
- Work with security experts and enforce strong authentication and authorization policies to mitigate security risks. CDEs present an additional, high-value attack vector, as they become the pipeline through which intellectual property flows.

Sample Vendors

Amazon Web Services; Codeanywhere; Coder; GitHub; Gitpod; Google; JetBrains Space; Red Hat; Replit; Strong Network

Gartner Recommended Reading

Quick Answer: How to Create a Frictionless Onboarding Experience for Software Engineers

Infographic: Platforms and Tools to Scale the Delivery of High-Quality Software

Innovation Insight for Internal Developer Portals

Innovation Insight for ML-Powered Coding Assistants

Innersource

Analysis By: Arun Chandrasekaran, Mark O'Neill, Arun Batchu, Anne Thomas, Mark Driver

Benefit Rating: Moderate

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

Innersource is the practice of bringing the open-source software principles of collaboration and openness into the organization for any form of software development.

Why This Is Important

Innersource can bring the benefits of collaborative, agile and robust software development principles to organizations. Every company is a software company, and applying innersource principles allows them to safely emulate the open-source model of collaborative development. Innersource can be a safe way to experiment with open-source practices while enabling governance at scale.

Business Impact

- Innersource practices lead to a culture centered on transparency and collaboration, which creates a healthy working environment.
- Innersource encourages professionals to think about software usage strategically rather than trying to solve tactical problems.
- Organizations can use innersource to align with external open-source communities and foundations to tap into their knowledge.
- Embracing open-source principles can encourage employee retention through the recognition of contributions.

Drivers

- The need for communication and collaboration between disparate software development teams within organizations drives the need for innersource.
- Software developers are now very familiar with open-source practices. They have been interacting with code repositories, using well-known open-source tools and even, in some cases, contributing to open-source projects. This creates the foundation for the effective application of open-source principles inside organizations.
- Code quality and velocity can be improved through the usage of an automated quality and consistency checking process. Such a process is often part of the technical infrastructure of an innersource effort, many times via the work of an open-source program office (OSPO).
- The increased reuse of functionality — such as code libraries or via APIs — are also key drivers of innersource initiatives.
- An important reason to adopt innersource is to increase knowledge sharing and potentially create a full-stack view through the democratization of know-how.

Obstacles

- Obtaining sponsorship from senior management can be challenging due to the long-term outlook and cultural transformation required for a successful implementation of innersource.
- Governance of innersource is a challenge since it may seem chaotic and fast-moving at first.
- Many organizations do not yet have self-service access to deploy code, or access APIs or shared libraries.
- Adoption of innersource requires a high degree of infrastructure automation, which can be challenging.
- Cultural resistance to innersource can inhibit broader adoption and maturity.

User Recommendations

- Encourage teams to open their code repositories for others to contribute to them by creating good documentation and contribution guidelines.
- Create smaller and flatter product teams that embrace DevOps, with an emphasis on egalitarian decision making and constant communication on product direction.
- Communicate and templatize best practices on coding standards, an API style guide, preferred tooling and common repositories, and a code release process.
- Enable self-service for infrastructure as a service (IaaS), and build automation, test and release pipelines.
- Create an innersource portal as a focal point for innersource initiatives.
- Explore gamification as a way to recognize and reward developers who contribute to innersource initiatives. Badges and recognition will improve employee motivation.
- Conduct hackathons to stimulate the creative process, and encourage collaboration across teams and codebases. This will also break down the silo mentality.

Gartner Recommended Reading

A CTO’s Guide to Open-Source Software: Answering the Top 10 FAQs

2023 Planning Guide for Application Development

Cultivate a Culture of Ownership in Your Software Engineering Organization

Video Spotlight: Unlocking the Value of Open Source at Fannie Mae

Best Practices for Setting Up an Open-Source Program Office

Observability-Driven Development

Analysis By: Manjunath Bhat

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Emerging

Definition:

Observability-driven development (ODD) is a software engineering practice that provides fine-grained visibility and context into system state and behavior by designing observable systems. ODD works by instrumenting code to unravel a system's internal state with externally observable data. As part of a shift-left approach to software development, ODD makes it easier to detect, diagnose and resolve unexpected anomalies early in the development life cycle.

Why This Is Important

Building observable systems can expedite issue resolution as observability data is a useful debugging aid. Designing for observability also amplifies the benefits of other resilience engineering practices, such as site reliability engineering (SRE) and chaos engineering. ODD enables software engineers and product owners to understand how the software is performing and how it is being used. However, the practices to institutionalize ODD are still emerging.

Business Impact

Built-in observability helps develop reliable software. ODD reduces the time to troubleshoot issues in production and preproduction environments. This feature helps software engineering teams reduce cycle time, making developers more productive during testing. ODD also helps ship code confidently since observable data makes it easier to troubleshoot production issues, thus minimizing downtime. In business terms, it translates to compliance with regulatory and contractual SLAs.

Drivers

- Organizations adopting SRE and chaos engineering practices need the data provided by observability design. These practices are fundamentally data-driven and support software reliability.
- User experience (UX) is subjective and difficult to measure. It depends on various factors that span the complete technology stack. Some factors, such as last-mile network connectivity, affect UX but are difficult to optimize unless systems are designed to be observable and extract data related to such signals, like response rates and latency.
- Mobile and edge environments present unforeseeable challenges since applications can run on potentially unknown devices and untrusted environments. These production environments can significantly differ from the local environments used to test the application. Therefore, ODD can help capture and investigate the “unknown unknowns” at runtime.
- Distributed systems exhibit emergent behavior and are difficult to predict. Therefore, the need for observability increases as issues can arise due to unexpected component interactions. Troubleshooting issues in distributed applications requires fundamentally different techniques than monolithic or client and server applications. Building observability narrows down the problem domain and helps engineers inspect the problematic component.
- OpenTelemetry is an open standard that has seen increased open-source implementations and vendor adoption over the past year. This has improved consistency when adopting ODD across products. Organizations also favor using open standards and protocols to ingest telemetry data. This makes ODD accessible to most developers even when they lack the budget for commercial tools.

Obstacles

- Monitoring and, by association, observability, is commonly viewed as an operational responsibility. Software engineers often lack expertise with observability as a practice and with the tools and frameworks used to implement observability.
- Software engineering leaders must overcome the perception that observability can be achieved merely by implementing an observability tool. Observability is a domain that must be designed and built into systems to ensure that it provides business benefits.
- A piecemeal approach to observability may thwart efforts to adopt ODD at scale. As a standard practice, ODD provides greater benefits when all system components are designed with an observability mindset. For example, distributed tracing requires that all components contributing to the trace are "instrumented" and propagate context for diagnosing response-time issues. Platform teams entrusted with driving ODD at scale can help overcome this obstacle.

User Recommendations

- Adopt ODD as a standard software engineering practice to handle unexpected and unforeseeable system behaviors and anomalies.
- Make observability a priority since it provides critical insights to resolve production errors. It provides continual feedback to understand how the software is being used.
- Keep up with the pace of innovation in observability by using open standards and open-source technologies, such as OpenTelemetry.
- Be wary of vendor hype regarding observability merely to provide access to logs, metrics and traces. Use ODD as a fundamental software engineering practice that improves UX and resilience with granular insight into system state and behavior.

Sample Vendors

Chronosphere; Datadog; Dynatrace; Grafana; Honeycomb; Logz.io; New Relic; observIQ; ServiceNow; Splunk

Gartner Recommended Reading

How to Identify and Resolve Application Performance Problems

Improve Software Quality by Building Digital Immunity

Software Engineering Intelligence Platforms

Analysis By: Frank O'Connor

Benefit Rating: High

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

Software engineering intelligence (SEI) platforms shine a light on engineering data. They provide software engineering leaders and teams with insights into what the engineering data really says about the development and deployment of software products and the effectiveness of software engineering teams.

Why This Is Important

SEI platforms help teams better understand how software solutions are being built and deployed. Teams can easily see where they spend time and how they approach code quality (such as code reviews), better understanding the team flow through key metrics like deployment frequency and cycle time. SEI platforms provide software engineering leaders with a single, open and transparent solution for reporting key engineering metrics across four key categories, namely, velocity and flow, business alignment, quality, and operations effectiveness.

Business Impact

Business impacts of SEI platforms are:

- **Evidence-based storytelling:** SEI platforms collect quantitative data from engineering systems to measure and track progress, readily proving the value created to business stakeholders.
- **Increased productivity:** Development teams can use engineering data to find inefficiencies in the plan, build and deploy cycle.
- **Better business alignment:** By tracking planned activities against completed ones, teams can improve estimations and more readily show how much time they spend on the high-value features of the business.

Drivers

- **The data challenge:** The proliferation of specialist roles and related tooling, which has become standard in modern software engineering organizations, makes it more difficult to collect engineering data and to generate insights that drive improved business outcomes. SEI platforms ameliorate this challenge by providing rich integrations for collecting data for many of the popular systems in use in engineering organizations.
- **Growing need for evidence-based measures of value:** Software engineering leaders are under increasing pressure to use data to first articulate, and then, prove the value that software engineering teams deliver.
- **Rising popularity of measurement frameworks:** Measurement frameworks, such as DevOps Research and Assessment (DORA) and Satisfaction, Performance, Activity, Communication and collaboration, Efficiency and flow (SPACE), are increasing in popularity among the software engineering community, as they can be easily implemented and consistently deployed using SEI platforms.
- **Benchmarking:** Many organizations desire to know how their engineering organization compares to similar-size engineering organizations. SEI platforms serve clients from different industries and can build industry-level benchmarks and comparisons.
- **Need for higher productivity:** Over the last 18 months Gartner has observed increased client interest on the topic of developer productivity, with questions concerning “how to do more with less,” and “what to measure to track and improve developer productivity.” SEI platforms provide teams with better visibility of their daily activities, allowing them to remove bottlenecks and be more productive.

Obstacles

- **Data confidentiality and security:** SEI platforms need access to sensitive and often proprietary data. Organizations may not wish to give a third party access to sensitive data.
- **Integration effort:** SEI platforms may struggle to collect data from homegrown or heavily customized software engineering tools. It may not be possible to collect data from these systems or the data collected may be difficult to analyze.
- **Perception of micromanagement:** Engineers may feel that the collected data will be used to spy on them. This might erode trust, resulting in the team productivity suffering and talented people possibly leaving the organization. SEI platforms can also be misused when software engineering leaders collect individual metrics instead of team-level metrics.
- **Organization maturity:** The effort and cost associated with adopting an SEI platform may not be warranted in organizations with immature engineering processes, or where data is not available for the delivery value stream.

User Recommendations

- Provide consistent visibility into team productivity and value delivery by using SEI platforms to surface leading indicators of successful business outcomes.
- Demonstrate the business value of software engineering teams by selecting SEI platforms that support the linking of desired strategic business outcomes with measurable tactical engineering metrics.
- Exercise caution when asked to provide comparisons, only comparing teams against other similar organizations or the wider industry by using the competitive benchmarking capabilities of SEI platforms. Avoid the pitfall of comparing teams or individuals within the engineering organization.

Sample Vendors

Allstacks; BlueOptima; Code Climate; Faros AI; Haystack; Jellyfish; LinearB; Logilica; Oobeya; Sleuth

Gartner Recommended Reading

Innovation Insight for Software Engineering Intelligence Platforms

How to Measure and Improve Developer Productivity

Value Stream Management Platforms

Analysis By: Hassan Ennaciri, Akis Sklavounakis

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Early mainstream

Definition:

A value stream management platform (VSMP) is a platform that seeks to optimize end-to-end product delivery and improve business outcomes. VSMPs are typically tool-agnostic. They connect to existing tools and ingest data from all phases of software product delivery — from customers’ needs to value delivery. VSMPs help software engineering leaders identify and quantify opportunities to improve software product performance by optimizing cost, operating models, technology and processes.

Why This Is Important

As organizations scale their agile and DevOps practices, higher-level metrics that assess performance and efficiency of their product delivery are essential. VSMPs integrate with multiple data sources to provide DevOps-related telemetry. These insights enable stakeholders to make data-driven decisions in an agile manner and correct course as needed. The visualization capabilities of VSMPs help product teams analyze customer value metrics against the cost required to deliver that value.

Business Impact

VSMPs help organizations bridge the gap between business and IT by enabling stakeholders to align their priorities to focus on delivering customer value. VSMPs can provide CxOs with strategic views of product delivery health and pipelines, allowing them to make data-driven decisions about future product investments. These platforms also provide product teams with end-to-end visibility and insight into the flow of work to help them address constraints and improve delivery.

Drivers

- Improved software delivery with business priorities and objectives.
- Timely decision making driven by insights from data.
- Optimization of delivery flow through reduction of waste and elimination of bottlenecks.
- Visibility and mapping of end-to-end software delivery processes and identification of cross-team dependencies.
- Quality and velocity improvements of product deployments.
- More stringent governance, security and compliance requirements.

Obstacles

- VSMPs are not focused on continuous integration/continuous delivery (CI/CD) capabilities. Execution of the delivery pipeline requires use of a custom toolchain or DevOps platform.
- VSMPs require customization and data from tools used by multiple stakeholders in the organization, sometimes outside of software delivery. Collaboration with these key stakeholders to deliver the desired insights is paramount.
- VSMPs are still evolving and not all vendors have all the core capabilities.

User Recommendations

- Accelerate business outcomes by leveraging real-time, data-driven metrics and value stream insights provided by VSMPs.
- Leverage VSMPs' AI-powered analytics and insights to surface constraints, detect bottlenecks and improve flow.
- Build customized dashboards and views of product delivery for multiple stakeholders and leadership.
- Utilize VSMPs to assess the performance, quality and value of products, including development costs and ROI.
- Use VSMPs to gain a consolidated view of governance, security and compliance across all product lines.

Sample Vendors

Broadcom; ConnectALL; Digital.ai; HCLSoftware; IBM; OpenText; Opsera; Planview; Plutora; ServiceNow

Gartner Recommended Reading

Market Guide for Value Stream Management Platforms

Tools for Delivering Business Metrics to Software Engineering Teams

Market Guide for Value Stream Delivery Platforms

Use the Right Metrics in the Right Way for Enterprise Agile Delivery

Internal Developer Portal

Analysis By: Manjunath Bhat

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Internal developer portals serve as the interface that enables self-service discovery and access to resources in complex, cloud-native software development environments. They can include software catalogs, scorecards to benchmark software quality, scaffold templates, product documentation, plug-ins for extensibility and automation workflows. Developer portals help improve developer productivity, operational efficiency and enhance governance by providing shared visibility across multiple teams.

Why This Is Important

Internal developer portals help software developers navigate infrastructure complexity, understand service interdependencies and enable faster release cadence in at least three ways. First, they serve as a common viewpoint for multiple teams of developers. Second, they provide developers with self-service access to underlying platform components and environments. Third, they provide a centralized place to score applications and measure progress against reliability and security requirements.

Business Impact

Developer portals can have the following business impacts:

- Developer experience and productivity: Help development teams improve their delivery cadence by improving developer experience, reducing cognitive load and shortening the onboarding time.
- Reliability and resilience: Aim to provide visibility to application health and include scorecards to assess their production readiness.
- Security and governance: Include prebuilt toolkits, templates and curated libraries that help create “paved roads” with built-in compliance, security and audit policies.

Drivers

- Platform engineering: Organizations are adopting platform engineering principles and creating platform teams to scale cross-cutting capabilities across multiple development teams. Platform teams curate internal developer platforms to abstract away the complexity of siloed systems and processes. Internal developer portals serve as an interface through which developers can consume the capabilities of internal developer platforms.
- Backstage: Backstage is one of the first open-source frameworks for building developer portals. It was created at Spotify and is now a Cloud Native Computing Foundation (CNCF) incubating project. The thriving open-source community supporting Backstage has largely contributed to its enormous mind share and rapid adoption. Hundreds of organizations have adopted Backstage since it was open-sourced in 2020. Backstage’s success continues to drive interest, momentum and competition in this space.
- Developer experience: With software at the core of all digital innovation today, a great developer experience that accelerates software development becomes a key competitive advantage. Therefore, software engineering leaders are increasingly focused on minimizing developer friction and frustration. The ability to curate and provide customizable, developer-friendly experiences within the developer portal and reign in complexity will drive their appeal for both product and platform teams.

- Innersource: To enable rapid innovation and facilitate greater collaboration and knowledge sharing, software engineering leaders are adopting innersource approaches to software development. However, innersource requires an easy way for other teams to discover and search for existing projects within their organization. This is why organizations adopting innersource are turning to internal developer portals to make projects available and discoverable by other teams. See InnerSource Portal.

Obstacles

- Prerequisites: The successful adoption of internal developer portals goes beyond deploying a tool and requires certain prerequisites to be in place. For example, application services and their dependencies must be organized with consistently defined metadata that helps track their usage, performance and team ownership.
- Absence of platform teams: A dedicated platform team to manage and evolve the portal as a product is necessary to ensure the portal meets desired objectives. The absence of a dedicated platform team, and more so, led by a platform product owner to manage the portal as a product results in a disconnect between developer expectations and the portal's capabilities.
- Lack of developer buy-in: Although the developer portal serves as the "window" to the underlying platform's capabilities, it should provide "paved roads" and not "forced marches" — portal use should remain the choice of the development team. Trying to force development workflows into organizationwide blueprints for building developer portals without involving developers is a recipe for failure.

User Recommendations

- Use internal developer portals to scale cross-cutting software engineering capabilities across multiple development teams and streamline the software delivery life cycle.
- Do not assume that internal developer portals are turnkey solutions — they require a lot of prerequisites to be in place and many cases involve several weeks of prework activities. For example, Backstage requires codification of service-related metadata in YAML files before the content shows up in the software catalog.
- Ensure that the platform team includes internal developer portals in their charter. Continuously innovate portal capabilities by appointing a platform product owner for the developer portal to manage its roadmap, gather feedback and market its capabilities.

Sample Vendors

Atlassian; Calibo; CodeNOW; configure8; Cortex; Mia-Platform; OpsLevel; Port; Roadie

Gartner Recommended Reading

Innovation Insight for Internal Developer Portals

Drive Innovation by Enabling Innersource

Adopt Platform Engineering to Improve the Developer Experience

Cool Vendors in Platform Engineering for Improving Developer Experience

Event-Driven APIs

Analysis By: Max van den Berk

Benefit Rating: Moderate

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

Event-driven APIs describe the interfaces used in event-driven architecture, including subscriptions and channel creation. Event-driven APIs differ from traditional request/response REST APIs because they enable asynchronous communication. Popular standards for event-driven APIs include AsyncAPI.

Why This Is Important

Event-driven APIs differ from traditional request/response APIs because they enable real-time notification of changes to data and application state. They are also provided as part of an event broker infrastructure.

Industries like financial services require timely updates of data and application state, which have driven the usage of event-driven architecture and event-driven APIs.

Integration scenarios also drive the usage of event-driven APIs, including data synchronization across SaaS services.

Business Impact

Event-driven APIs open up business opportunities for faster response to change and streaming analytics.

They also facilitate, and help standardize, event-driven architecture (EDA). Event-driven APIs bring advantages such as enabling push notifications, which are much more efficient and less expensive (from both a time and networking perspective) than polling.

Drivers

- Mobile applications involve the widespread use of events, such as webhook notifications using server-sent events (SSEs) delivered over HTTP.
- Data analytics and artificial intelligence drive new use cases related to events and notifications, which in turn drive the need for event-based APIs.
- Open-source and cloud event broker technologies, especially Apache Kafka, have raised awareness of publish-subscribe infrastructure, as well as accessibility to it, and have encouraged adoption of EDA.
- The OpenAPI Specification (OAS) version 3, introduced in 2017 and now reaching widespread adoption as a standard for publishing APIs, includes a provision for callbacks as a means to describe event-driven APIs. OAS version 3.1 (February 2021) adds support for webhooks, the most common event-driven API approach for internet-facing APIs. It should be noted, however, that webhooks are a one-to-one pattern, as opposed to EDA, which also supports a many-to-many pattern.
- AsyncAPI, which defines a standard for how event-driven APIs are published, is supported by the Linux Foundation, which is the same standards body that houses OAS.

Obstacles

- API mediation products, such as API gateways, are often built or configured only for request-response APIs. In some cases, the patterns of event-driven APIs (including fire-and-forget or publish-and-subscribe methods) are not supported in the API mediation product.
- Protocol support for event-driven APIs is diverse, including webhook (HTTP/S), WebSockets, MQTT, advanced message queuing protocol (AMQP) and Apache Kafka. This makes decision making more difficult for software engineering leaders.
- Although API portals are widely used for publishing request/response APIs, their usage for event-driven APIs is nascent.
- Reuse of schema definitions, such as Protobuf, Avro and graphQL, is not standardized, and might require tooling from the end user to support it.
- Event-driven APIs, like all event-driven architecture, introduce challenges for debugging and testing.

User Recommendations

- Evaluate whether your chosen API mediation products, including API gateways as part of API management, support event-driven APIs as part of your technology selection process.
- Adopt a community-of-practice approach to raise your organization's overall skills on event-driven APIs, and their relationship to event-driven architecture and stream processing and analytics. This may be a specific EDA community of practice, or part of an API community of practice.
- Identify opportunities to replace request/response APIs with event-driven APIs within your API portfolio to enable both internal and external integration scenarios.

Sample Vendors

Ably; Axual; Axway; Confluent; Gravitee; Postman; SmartBear Software; Software AG; Solace; TIBCO Software

Gartner Recommended Reading

Using Event-Driven Integration With Enterprise Applications

Maturity Model for Event-Driven Architecture

At the Peak

AI Coding Assistants

Analysis By: Arun Batchu, Philip Walsh

Benefit Rating: Transformational

Market Penetration: 5% to 20% of target audience

Maturity: Emerging

Definition:

AI code assistants help developers by leveraging foundation models trained on millions of lines of code. Developers prompt the assistants with natural language, code snippets and triggers to generate code blocks. They also analyze, explain and refactor code; generate documentation; and translate between programming languages. Assistants support multiple languages and integrate into programming environments like code editors, command-line and chat.

Why This Is Important

Software demand exceeds most organizations' capacity. Existing developers are maxed out, unable to build features fast enough or find satisfaction in their work. Recruiting more developers is difficult, as is upskilling staff. AI coding assistants are emerging as accelerators, boosting developer productivity and happiness. By handling routine tasks, the assistants enable developers to focus on higher-value activities. This allows organizations to deliver more features faster with existing teams.

Business Impact

AI coding assistants boost organizations' software development capabilities by:

- Reducing code typing through code recommendations that developers can accept
- Decreasing delivery times by suggesting solutions faster than humans
- Increasing time spent on creative work, as assistants handle repetitive coding
- Improving developer job satisfaction by allowing them to focus on high-value tasks
- Helping developers rapidly learn new languages/frameworks and hone skills

Drivers

- Advances in foundational large language models (LLMs) are accelerating the capability of coding assistants.
- AI coding assistants, previously available only to developers in technology companies, are now accessible to all.
- Enterprises, desperate for ways to increase the productivity of highly paid developers, are able to justify the cost of rolling out these tools to all their development staff.
- Developers are rapidly adopting these tools in their personal projects, as quickly as their employers make them available.
- Vendors of these assistants are increasing, with the hope of monetizing the demand.
- Competition is driving rapid innovation, fueling the adoption rate.
- Non-computer-science or software engineering majors are rapidly gaining software development skills, opening up new talent pools that otherwise were not available for enterprises.

Obstacles

- Most training data is from open domains like open source, which may contain problematic code with security, IP, toxicity and bias issues that could be learned..
- Vendors use prompts and conversations to improve models, so those trained on proprietary data could expose it to harmful third-party use.
- Pending lawsuits against some vendors may decelerate or stop adoption of not only the affected vendors, but the entire market.
- Future AI regulations might force technological compromises that decrease the effectiveness of these models.
- Regional AI regulations might prevent organizations from adopting these tools.
- Productivity gains in coding activities might get dwarfed by the overall inefficiency of an organization’s software development life cycle.
- Lack of proper training to maximize these tools might dampen the return on investing in them.

- Trusting the generated code and text without verification may lead to production quality problems.

User Recommendations

Software engineering leaders should:

- Trial the tools and toolchain to catch downstream errors by forming a pilot team of engineers, testers, security staff and compliance experts.
- Narrow down the vendor choices by determining if forcing functions like private network needs, permissive licensing requirements, or enterprise code context apply.
- Scale deployment by developing a rollout plan.
- Remove bottlenecks from the SDLC value stream to maximize ROI on coding productivity gains by using software engineering intelligence tools.
- Achieve prompt engineering consistency and quality across the enterprise by creating a small, highly skilled platform team that develops and publishes prompt engineering patterns and best practices.
- Develop a learning and development plan for developers to master the tools.
- Maintain flexibility by being prepared to switch vendors in the short term, in case it becomes untenable to use your primary vendor.

Sample Vendors

Amazon Web Services; Codeium; GitHub; Google; Hugging Face; IBM; OpenAI; Replit; Sourcegraph; Tabnine

Gartner Recommended Reading

Innovation Insight for ML-Powered Coding Assistants

Assessing How Generative AI Can Improve Developer Experience

Quick Answer: Should Software Engineering Teams Use ChatGPT to Generate Code?

Quick Answer: How Can Generative AI Be Used to Improve Testing Activities?

Quick Answer: Can We Use ChatGPT for Code Transformation and Modernization?

Open-Source Program Office

Analysis By: Nitish Tyagi, Mark O'Neill, Mark Driver, Arun Chandrasekaran

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Early mainstream

Definition:

An open-source program office (OSPO) is the center of competency to build strategies for governing, managing, promoting and efficiently using open-source software (OSS) and open-source data or models. The OSPO includes members from application delivery, legal, IT security, procurement and product management.

Why This Is Important

OSS is the backbone of digital innovation, with more than 95% organizations using it. However, ad hoc usage of OSS can lead to legal, security and viability risks. To manage these risks, OSPO or similar programs adoption has increased by 50% (see TODO's OSPO Survey 2022). Many large enterprises such as Alibaba, Apple, Bosch and Capital One have done so. OSPOs build cohesive strategies to efficiently use and promote OSS (see A CTO's Guide to Open-Source Software: Answering the Top 10 FAQs).

Business Impact

OSS drives accelerated software development, innovation, flexibility, cost savings and talent retention. A well-run OSPO ensures efficient consumption of OSS, implements effective governance policies, facilitates contributions and communicates the value of OSS to stakeholders. Embracing OSS, particularly contributing to or maintaining an OSS project, positively affects the retention of employees, especially developers. As the OSPO matures, it becomes a strategic partner in all technology decisions.

Drivers

- OSS is ubiquitous, but enterprises with an ad hoc approach to OSS have limited visibility into where and how OSS is used within their technology stack. An OSPO provides a strategic approach and required visibility by using correct tools.
- Poorly governed use of OSS, without proper assessment, exposes an enterprise to security, legal and viability risks. For example, ungoverned use of OSS components may violate licensing terms or infringe upon intellectual property rights, or OSS with a weak community may incur huge technical debt. An OSPO is responsible for governing the use of OSS.
- Software developers wish to contribute to open-source projects that they use in their day-to-day work. Actively supporting contributions to OSS projects helps talent acquisition and retention and ensures the longevity and relevance of the OSS your organization uses.
- Establishing the source and provenance of software components is essential, as legal requirements for software bills of materials grow (see Executive Order on Improving the Nation's Cybersecurity).

Obstacles

- The lack of funding and executive sponsorship is the biggest reason for the limited adoption of OSPOs. Establishing the correct metrics to predict the success of an OSPO is often challenging for organizations.
- Silos between different teams make it difficult for an OSPO to drive collaboration.
- Total cost of ownership is always attached with using an OSS project, but costs are often ignored when organizations do not plan their use of OSS.
- A lack of self-service tooling and automation in applying the policies related to consumption and publication can delay the software development life cycle.

User Recommendations

- Establish an enterprisewide OSPO and shift from ad hoc use of OSS to strategic use of OSS by building policies and processes to govern, assess and manage OSS. Fill up the core OSPO roles and include members from different stakeholders groups, such as application delivery, security, legal and procurement.
- Define the correct set of metrics such as developer productivity, and hiring and retention rates to measure the OSPO's success. Formulate and enforce a governance policy for consumption, contribution and creation of OSS by building an OSS governance committee.
- Work with platform engineering teams to provide the correct set of tools for artifact repository, security, issue tracking, continuous integration, continuous development, collaboration and knowledge management.
- Evaluate your OSPO's maturity level and execute strategies to gradually promote it to the next level as appropriate.

Sample Vendors

Bitergia; Cloud Native Computing Foundation (CNCF); Linux Foundation; TODO Group

Gartner Recommended Reading

Best Practices for Setting Up an Open-Source Program Office

A CTO's Guide to Open-Source Software: Answering the Top 10 FAQs

How to Create and Enforce a Governance Policy for Open-Source Software

How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks

Video Spotlight: Unlocking the Value of Open Source at Fannie Mae

Prompt Engineering

Analysis By: Frances Karamouzis, Afraz Jaffri, Jim Hare, Arun Chandrasekaran, Van Baker

Benefit Rating: High

Market Penetration: 1% to 5% of target audience

Maturity: Emerging

Definition:

Prompt engineering is the discipline of providing inputs, in the form of text or images, to generative AI models to specify and confine the set of responses the model can produce. The inputs prompt a set that produces a desired outcome without updating the actual weights of the model (as done with fine-tuning). Prompt engineering is also referred to as "in-context learning," where examples are provided to further guide the model.

Why This Is Important

Prompt engineering is the linchpin to business alignment for desired outcomes. Prompt engineering is important because large language models (LLMs) and generative AI models in general are extremely sensitive to nuances and small variations in input. A slight tweak can change an incorrect answer to one that is usable as an output. Each model has its own sensitivity level, and the discipline of prompt engineering is to uncover the sensitivity through iterative testing and evaluation.

Business Impact

Prompt engineering has the following business impacts:

- **Performance:** It helps improve model performance and reduce hallucinations.
- **Business alignment**: It allows subject data scientists, subject matter experts and software engineers to steer foundation models, which are general-purpose in nature, to align to the business, domain and industry.
- **Efficiency and effectiveness:** Alternative options, such as building a model from scratch or fine-tuning, can be much more complex, drive longer time to market and be more expensive.

Drivers

- **Balance and efficiency:** The fundamental driver for prompt engineering is it allows organizations to strike a balance between consuming an “as is” offering versus pursuing a more expensive and time-consuming approach of fine-tuning. Generative AI models, and in particular LLMs, are pretrained, so the data that enterprises want to use with these models cannot be added to the training set. Instead, prompts can be used to feed content to the model with an instruction to carry out a function.

- **Process or task-specific customizations or new use cases:** The insertion of context and patterns that a model uses to influence the output generated allows for customizations for a particular enterprise or domain, or regulatory items. Prompts are created to help improve the quality for different use cases — such as domain-specific question answering, summarization, categorization, and so on — with or without the need for fine-tuning a model, which can be expensive or impractical. This would also apply to creating and designing new use cases that utilize the model’s capability for image and text generation.

- **Validation and verification:** It is important to test, understand and document the limits and weaknesses of the models to ensure a reduced risk of hallucination and unwanted outputs.

Obstacles

- **Embryonic nature of the discipline:** Prompt engineering processes and roles are either unknown or enterprises have a low level of understanding and experience. Gartner webinar polling data (over 2,500 responses; see Executive Pulse: AI Investment Gets a Boost From ChatGPT Hype) revealed that approximately 60% of respondents self-reported that they had not heard of prompt engineering. And 90% of those same respondents revealed that their organization did not currently have prompt engineers.
- **Role alignment:** Data scientists are critical to understanding the capabilities and limits of models, and to determine whether to pursue a purely prompt-based or fine-tuning-based approach (or combination of approaches) for customization. The ultimate goal is to use machine learning itself to generate the best prompts and achieve automated prompt optimization. This is in contrast to an end user of an LLM who concentrates on prompt design to manually alter prompts to give better responses.
- **Lack of business alignment:** There is often a lack of consensus on prompt engineering's business approach, as well as agreed-upon standards, methodology and approaches. This has led to fierce debates on the value of prompt engineering and how to establish governance.
- **Risk:** Beyond the early stages of awareness and understanding, the biggest obstacle may be that prompt engineering is focused on verification, validation, improvement and refinement; however, it's not without risk. Prompt engineering is not the panacea to all of the challenges. It helps to manage risk, not remove it completely. Errors may still occur, and potential liability is at stake.

User Recommendations

- Rapidly build awareness and understanding of prompt engineering in order to quickly start the journey of shape-shifting the appropriate prompt engineering discipline and teams.
- Build critical skills across a number of different team members that will synergistically contribute critical elements. For example, there are important roles for data scientists, business users, domain experts, software engineers and citizen developers.
- Communicate and cascade the message that prompt engineering is not foolproof. Rigor and diligence need to permeate and work across all the enterprise teams to ensure successful solutions.

Sample Vendors

FlowGPT; HoneyHive; LangChain; PromptBase; Prompt Flow; PromptLayer

Gartner Recommended Reading

Quick Answer: How Will Prompt Engineering Impact the Work of Data Scientists?

Quick Answer: What Impact Will Generative AI Have on Search?

Accelerate Adoption of Generative AI by Offering an FMOps- or a Domain-Specific Partner Ecosystem

Glossary of Terms for Generative AI and Large Language Models

Design Systems

Analysis By: Will Grant, Brent Stewart

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Design systems are collections of reusable assets that are based on clear visual, user interfaces and technical standards. They serve as building blocks to quickly and consistently design and develop digital products. Organizations can deliver better user experience for customers and reduce development effort, while preparing to leverage emerging generative AI technologies. A complete design system is composed of: style assets, structural assets, code components and documentation.

Why This Is Important

Using a design system is one of the most effective ways of ensuring design consistency across digital product offerings. Building a design system into your software development process contributes to increased brand consistency, better user experiences and higher front-end developer productivity. Popular SaaS platforms — including Salesforce and SAP — maintain their own design systems to facilitate application design and development on their platforms.

Business Impact

A design system is one of the most important strategic assets for an organization that builds and configures digital products. A robust design system that is resourced and maintained well will drastically shorten design and development timelines. It will ensure the user interface (UI) design is consistent, predictable and usable, and guarantee brand compliance across an organization's full portfolio of digital products both customer and employee facing.

Drivers

- **Speed:** Design systems drastically reduce the time required to design and code the presentation layer of software by reducing the need to repeatedly design from a blank template. Design systems enable easy component assembly and fast screen design tweaks that allow designers to work at pace.

- **Usability:** Design systems are typically composed of time-tested, proven UI design patterns that are familiar to most users. Foundational UI design heuristics, such as "visibility of system status" or "recognition rather than recall" are built into these patterns.

- **Consistency:** Design systems enable the creation of consistent user experiences across disparate teams, whether they are feature teams delivering into a single product or multiple product teams sharing a design system across a larger product portfolio.

- **Scale:** Design systems make it easy for designers and developers to share common, approved design assets and code components across an entire portfolio of digital products, and to work independently using the same assets.

- **Reduction of defects:** Over time, design system code components become "hardened," leading to far fewer production defects in the presentation layer.

- **Brand compliance:** Design systems reinforce a brand identity and infuse key elements such as color and typography into every single design and code asset.

- **Accessibility:** Design system assets can be created in compliance with the latest Web Content Accessibility Guidelines (WCAG), eliminating unnecessary rework downstream.

- **AI-Ready:** Looking forward to emerging generative AI tools, a codified design system will be essential to enable generative AI to work with your in-house style.

Obstacles

- **Effort to create and maintain:** While design systems bring many benefits, they need to be resourced and maintained like any other internal product.
- **Lack of governance**: Without a clear process to update and encourage the use of a design system, it quickly becomes outdated and less impactful.
- **Cross-discipline buy-in**: Without the whole software engineering team getting behind a design system, there's a risk that *several* ad-hoc design systems emerge, multiplying effort and reducing impact.
- **Executive buy-in:** Few executive leaders outside of the design field are aware of the strategic importance and tremendous business value of design systems. Without strong leadership support, design systems easily become underutilized and diminished in terms of the value they add.
- **Originality:** Many UX practitioners are concerned that their designs will become too uniform and lack originality. This fear is generally unfounded as a well-implemented design system will still look and feel unique, while enhancing usability.

User Recommendations

- Conduct a regular review of available design systems by auditing leading examples.
- Assemble a team including UX, product development and product marketing to gather, organize, define and launch an enterprisewide design system, like an internal product.
- Don't start a new design system from scratch unless you're prepared for a significant multiyear investment to catch up with established systems.
- Update design and development processes to mandate the use of the design system rather than starting from scratch for new initiatives.
- Document your design system with style guides, technical component documentation, usage guides and accessibility considerations.
- Set up a structure of governance and stewardship to treat the design system as a vital internal product, with a backlog, roadmap and dedicated resource.
- Consider design system solutions that support design tokens and reusable components to enable the operationalization of your design system, and its readiness for AI-augmented design tools in the future.

Sample Vendors

Figma; Google; IBM; Knapsack; Sketch; Storybook; UXPin; zeroheight

Gartner Recommended Reading

Start Using Design Systems, Accelerate Digital Product Delivery

Predicts 2022: Generative AI Is Poised to Revolutionize Digital Product Development

How Design, Development and Product Management Can Work Together Successfully

The 4 Secrets of Design-Led Companies

How to Build a User Experience Team

Responsible AI

Analysis By: Svetlana Sicular

Benefit Rating: Transformational

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Responsible artificial intelligence (AI) is an umbrella term for aspects of making appropriate business and ethical choices when adopting AI. These include business and societal value, risk, trust, transparency, fairness, bias mitigation, explainability, sustainability, accountability, safety, privacy, and regulatory compliance. Responsible AI encompasses organizational responsibilities and practices that ensure positive, accountable, and ethical AI development and operation.

Why This Is Important

Responsible AI has emerged as the key AI topic for Gartner clients. When AI replaces human decisions and generates brand-new artifacts, it amplifies both good and bad outcomes. Responsible AI enables the right outcomes by ensuring business value while mitigating risks. This requires a set of tools and approaches, including industry-specific methods, adopted by vendors and enterprises. More jurisdictions introduce new regulations that challenge organizations to respond in meaningful ways.

Business Impact

Responsible AI assumes accountability for AI development and use at the individual, organizational and societal levels. If AI governance is practiced by designated groups, responsible AI applies to everyone involved in the AI process. Responsible AI helps achieve fairness, even though biases are baked into the data; gain trust, although transparency and explainability methods are evolving; and ensure regulatory compliance, despite the AI's probabilistic nature.

Drivers

- Responsible AI means a deliberate approach in many directions at once. Data science's responsibility to deliver unbiased, trusted and ethical AI is just the tip of the iceberg. Responsible AI helps AI participants develop, implement, utilize and address the various drivers they face.
- Organizational driver assumes that AI's business value versus risk in regulatory, business and ethical constraints should be balanced, including employee reskilling and intellectual property protection.
- Societal driver includes resolving AI safety for societal well-being versus limiting human freedoms. Existing and pending legal guidelines and regulations, such as the EU's Artificial Intelligence Act, make responsible AI a necessity.
- Customer/citizen driver is based on fairness and ethics and requires resolving privacy versus convenience. Customers should exhibit readiness to give their data in exchange for benefits. Consumer and citizen protection regulations provide the necessary steps, but do not relieve organizations of deliberation specific to their constituents.
- With further AI adoption, the responsible AI framework is becoming more important and is better understood by vendors, buyers, society and legislators.
- AI affects all ways of life and touches all societal strata; hence, the responsible AI challenges are multifaceted and cannot be easily generalized. New problems constantly arise with rapidly evolving technologies and their uses, such as using OpenAI's ChatGPT or detecting deepfakes. Most organizations combine some of the drivers under the umbrella of responsible AI, namely, accountability, diversity, ethics, explainability, fairness, human centricity, operational responsibility, privacy, regulatory compliance, risk management, safety, transparency and trustworthiness.

Obstacles

- Poorly defined accountability for responsible AI makes it look good on paper but is ineffective in reality.
- Unawareness of AI's unintended consequences persists. Forty percent of organizations had an AI privacy breach or security incident. Many organizations turn to responsible AI only after they experience AI's negative effects, whereas prevention is easier and less stressful.
- Legislative challenges lead to efforts for regulatory compliance, while most AI regulations are still in draft. AI products' adoption of regulations for privacy and intellectual property makes it challenging for organizations to ensure compliance and avoid all possible liability risks.
- Rapidly evolving AI technologies, including tools for explainability, bias detection, privacy protection and some regulatory compliance, lull organizations into a false sense of responsibility, while mere technology is not enough. A disciplined AI ethics and governance approach is necessary, in addition to technology.

User Recommendations

- Publicize consistent approaches across all focus areas. The most typical areas of responsible AI in the enterprise are fairness, bias mitigation, ethics, risk management, privacy, sustainability and regulatory compliance.
- Designate a champion accountable for the responsible development and use of AI for each use case.
- Define model design and exploitation principles. Address responsible AI in all phases of model development and implementation cycles. Go for hard trade-off questions. Provide responsible AI training to personnel.
- Establish operationalize responsible AI principles. Ensure diversity of participants and the ease to voice AI concerns.
- Participate in industry or societal AI groups. Learn best practices and contribute your own, because everybody will benefit from this. Ensure policies account for the needs of any internal or external stakeholders.

Sample Vendors

Amazon; Arthur; Fiddler; Google; H2O.ai; IBM; Microsoft; Responsible AI Institute; TAZI.AI; TruEra

Gartner Recommended Reading

A Comprehensive Guide to Responsible AI

Expert Insight Video: What Is Responsible AI and Why Should You Care About It?

Best Practices for the Responsible Use of Natural Language Technologies

Activate Responsible AI Principles Using Human-Centered Design Techniques

How to Ensure Your Vendors Are Accountable for Governance of Responsible AI

Platform Engineering

Analysis By: Bill Blosen, Paul Delory

Benefit Rating: Transformational

Market Penetration: 20% to 50% of target audience

Maturity: Adolescent

Definition:

Platform engineering is the discipline of building and operating self-service developer platforms for software development and delivery. A platform is a layer of tools, automations and information maintained as products by a dedicated platform team, designed to support software developers or other engineers by abstracting underlying complexity. The goal of platform engineering is to optimize the developer experience and accelerate delivery of customer value.

Why This Is Important

Digital enterprises need to respond quickly to customer and internal demands; therefore, flexible, complex distributed software architectures have become popular. Software product teams struggle to focus on features due to this complexity, which results in poor developer experience. Platform engineering provides a self-service, curated set of tools, automations and information driven by developer priorities to accelerate value delivery in line with internal stakeholders, such as security and architecture.

Business Impact

Platform engineering empowers application teams to deliver software value faster. It removes the burden of underlying infrastructure construction and maintenance and increases teams' capacity to dedicate time to customer value and learning. It makes compliance and controls more consistent and simplifies the chaotic explosion of tools used to deliver software. Platform engineering also improves the developer experience, thus reducing employee frustration and attrition.

Drivers

- Scale: As more teams embrace modern software development practices and patterns, economies of scale are created, whereby there is enough value to justify creating a platform capability shared by multiple teams.
- Cognitive load: Adoption of modern, distributed architectural patterns and software delivery practices means that the process of getting software into production involves more tools, subsystems and moving parts than ever before. This places a burden on product teams to build a delivery system in addition to the actual software they are trying to produce.
- Need for increased speed and agility: The speed and agility of software delivery is critical to CIOs. As a result, software organizations are pursuing DevOps which is a tighter collaboration of infrastructure and operations (I&O) and development teams to drive shorter development cycles, faster delivery and increased deployment frequency. This will enable organizations to respond immediately to market changes, handle workload failures better and tap into new market opportunities. Platform engineering can drive this type of cross-team collaboration.
- Emerging platform construction tools: Many organizations have built their own platforms, but to date, these platforms have been homegrown, individual efforts tailored to the unique circumstances of the organizations that build them. Platforms generally have not been transferable to other companies or sometimes even to other teams within the same company. However, a new generation of platform-building tools is emerging to change that.
- Infrastructure modernization: During digital modernization, some forward-looking I&O teams embrace a new platform engineering role as a way to deliver more value, increasing their relevance to the business.

Obstacles

- Lack of skills: Platform engineering requires solid skills in software engineering, product management and modern infrastructure, all of which are in short supply.
- Platform engineering is easily misunderstood: Traditional models of mandated platforms with limited regard for developer experience can easily be relabeled and thus not achieve the true benefits of platform engineering.
- Outdated management/governance models: Many organizations still use request-based provisioning models. Those need to give way to a self-service, declarative model, with the primary focus being the effectiveness of the end users developing and operating solutions using the platform.
- Internal politics: There are many intraorganizational fights that could derail platform engineering. Product teams may resist giving up control of their customized toolchains. There might also be no appetite to improve the developer experience. Enterprises may also refuse to fund platform engineering without a clear ROI.

User Recommendations

- Start small with cloud-native workloads: Begin platform-building efforts with thinnest viable platforms for the infrastructure underneath cloud-native applications such as containers and Kubernetes.
- Embed security into platforms: Enable shift-left security within DevOps pipeline platforms, which will provide a compelling paved road to engineers.
- Don't expect to buy a complete platform: Any commercially available tool is unlikely to provide the entirety of the platform you need. Thus, the job of the platform team is to integrate the components necessary for the platform to meet your needs.
- Implement a developer portal as part of your platform: An internal developer portal (IDP) serves as the user interface that enables self-service discovery and access to internal developer platform capabilities. Consider Backstage open-source or other commercial tools. Note: "IDP" has multiple meanings in this context, as well as in the industry.

Gartner Recommended Reading

How to Start and Scale Your Platform Engineering Team

Guidance Framework for Implementing Cloud Platform Operations

Adopt Platform Engineering to Improve the Developer Experience

Innovation Insight for Internal Developer Portals

AI-Augmented Software Engineering

Analysis By: Arun Batchu, Hema Nair, Oleksandr Matvitskyy

Benefit Rating: Transformational

Market Penetration: 5% to 20% of target audience

Maturity: Emerging

Definition:

The use of artificial intelligence (AI) technologies (e.g., machine learning [ML] and natural language processing [NLP]) to help software engineers create, deliver and maintain applications is designated AI-augmented software engineering (AIASE). This is integrated with engineers' existing tools to provide real-time, intelligent feedback and suggestions.

Why This Is Important

Today's software development life cycle includes such routine and repetitive tasks as boilerplate functional and unit-test code and docstrings, which AIASE tools automate. AI-powered automation enables software engineers to focus their time, energy and creativity on such high-value activities as feature development. Emerging AI tools discover the configurations that meet operational goals. Software builders who use these tools remain productive and engaged, and they stay longer in their jobs.

Business Impact

AIASE accelerates application delivery and allocates software engineering capacity to business initiatives with high priority, complexity and uncertainty, helping quality teams develop self-healing tests and nonobvious code paths. These tools automatically generate test scenarios previously created manually by testers, and detect test scenarios often missed by test teams. AIASE tools detect issues with code security, consistency or maintainability and offer fixes.

Drivers

Demand drivers include:

- The increasing complexity of software systems to be engineered
- Increasing demand for developers to deliver high-quality code faster
- Increasing numbers of application development security attacks
- Optimizing operational costs

Technology solution drivers include:

- The application of AI models to prevent application vulnerabilities by detecting static code and runtime attack patterns
- The increasing impact of software development on business models
- The application of large language models to software code
- The application of deep-learning models to software operations

Obstacles

- Hype about the innovation has caused misunderstandings and unrealistic expectations about the benefits of AIASE.
- There is a lack of deep comprehension of generated artifacts.
- There is limited awareness about production-ready tools.
- Software engineers who fear job obsolescence have shown resistance.
- There is a lack of transparency and provenance of data used for model training.
- Uneven, fragmented solutions that automate only some of the tasks in the software development life cycle (SDLC).
- AI skills such as prompt engineering, training, tuning, maintaining and troubleshooting models.
- High model training and inference costs at scale.
- Intellectual property risks stemming from models trained on nonpermissive licensed code.
- Privacy concerns stemming from code, and associated proprietary data leaking as training data for AI models.
- Technical employees' fear of jobs being automated by AI.

User Recommendations

- Pilot, measure and roll out tools only if there are clear gains.
- Verify the maintainability of AI-generated artifacts, including executable requirements, code, tests and scripts.
- Track this rapidly evolving and highly impactful market to identify new products that minimize development toil and improve the experience of software engineers, such as those that ease security and site operations burden.
- Reassure software engineers that AIASE is an augmentation toolset for human engineers, not a replacement.
- Pick providers (including open-source vendors) that supply visibility to training data and transparency on how the model was trained.
- Establish the correct set of metrics, such as new release frequency and ROI, to measure the success of AIASE.

Sample Vendors

Akamas; Amazon Web Services; Diffblue; Google; IBM; Microsoft; OpenAI; SeaLights; Sedai; Snyk

Gartner Recommended Reading

Innovation Insight for ML-Powered Coding Assistants

Infographic: Artificial Intelligence Use-Case Prism for Software Development and Testing

Market Guide for AI-Augmented Software Testing Tools

Generative AI

Analysis By: Svetlana Sicular, Brian Burke

Benefit Rating: Transformational

Market Penetration: 1% to 5% of target audience

Maturity: Adolescent

Definition:

Generative AI technologies can generate new derived versions of content, strategies, designs and methods by learning from large repositories of original source content. Generative AI has profound business impacts, including on content discovery, creation, authenticity and regulations; automation of human work; and customer and employee experiences.

Why This Is Important

Generative AI exploration is accelerating, thanks to the popularity of Stable Diffusion, Midjourney, ChatGPT and large language models. End-user organizations in most industries aggressively experiment with generative AI. Technology vendors form generative AI groups to prioritize delivery of generative-AI-enabled applications and tools. Numerous startups have emerged in 2023 to innovate with generative AI, and we expect this to grow. Some governments are evaluating the impacts of generative AI and preparing to introduce regulations.

Business Impact

Most technology products and services will incorporate generative AI capabilities in the next 12 months, introducing conversational ways of creating and communicating with technologies, leading to their democratization. Generative AI will progress rapidly in industry verticals, scientific discovery and technology commercialization. Sadly, it will also become a security and societal threat when used for nefarious purposes. Responsible AI, trust and security will be necessary for safe exploitation of generative AI.

Drivers

- The hype around generative AI is accelerating. Currently, ChatGPT is the most hyped technology. It relies on generative foundation models, also called “transformers.”
- New foundation models and their new versions, sizes and capabilities are rapidly coming to market. Transformers keep making an impact on language, images, molecular design and computer code generation. They can combine concepts, attributes and styles, creating original images, video and art from a text description or translating audio to different voices and languages.
- Generative adversarial networks, variational autoencoders, autoregressive models and zero-/one-/few-shot learning have been rapidly improving generative modeling while reducing the need for training data.
- Machine learning (ML) and natural language processing platforms are adding generative AI capabilities for reusability of generative models, making them accessible to AI teams.
- Industry applications of generative AI are growing. In healthcare, generative AI creates medical images that depict disease development. In consumer goods, it generates catalogs. In e-commerce, it helps customers “try on” makeup and outfits. In manufacturing, quality inspection uses synthetic data. In semiconductors, generative AI accelerates chip design. Life sciences companies apply generative AI to speed up drug development. Generative AI helps innovate product development through digital twins. It helps create new materials targeting specific properties to optimize catalysts, agrochemicals, fragrances and flavors.
- Generative AI reaches creative work in marketing, design, music, architecture and content. Content creation and improvement in text, images, video and sound enable personalized copywriting, noise cancellation and visual effects in videoconferencing.
- Synthetic data draws enterprises’ attention by helping to augment scarce data, mitigate bias or preserve data privacy. It boosts the accuracy of brain tumor surgery.
- Generative AI will disrupt software coding. Combined with development automation techniques, it can automate up to 30% of the programmers’ work.

Obstacles

- Democratization of generative AI uncovers new ethical and societal concerns. Government regulations may hinder generative AI research. Governments are currently soliciting input on AI safety measures.
- Hallucinations, factual errors, bias, a black-box nature and inexperience with a full AI life cycle preclude the use of generative AI for critical use cases.
- Reproducing generative AI results and finding references for information produced by general-purpose LLMs will be challenging in the near term.
- Low awareness of generative AI among security professionals causes incidents that could undermine generative AI adoption.
- Some vendors will use generative AI terminology to sell subpar "generative AI" solutions.
- Generative AI can be used for many nefarious purposes. Full and accurate detection of generated content, such as deepfakes, will remain challenging or impossible.
- The compute resources for training large, general-purpose foundation models are heavy and not affordable to most enterprises.
- Sustainability concerns about high energy consumption for training generative models are rising.

User Recommendations

- Identify initial use cases where you can improve your solutions with generative AI by relying on purchased capabilities or partnering with specialists. Consult vendor roadmaps to avoid developing similar solutions in-house.
- Pilot ML-powered coding assistants, with an eye toward fast rollouts, to maximize developer productivity.
- Use synthetic data to accelerate the development cycle and lessen regulatory concerns.
- Quantify the advantages and limitations of generative AI. Supply generative AI guidelines, as it requires skills, funds and caution. Weigh technical capabilities with ethical factors. Beware of subpar offerings that exploit the current hype.
- Mitigate generative AI risks by working with legal, security and fraud experts. Technical, institutional and political interventions will be necessary to fight AI's adversarial impacts. Start with data security guidelines.
- Optimize the cost and efficiency of AI solutions by employing composite AI approaches to combine generative AI with other AI techniques.

Sample Vendors

Adobe; Amazon; Anthropic; Google; Grammarly; Hugging Face; Huma.AI; Microsoft; OpenAI; Schrödinger

Gartner Recommended Reading

Innovation Insight for Generative AI

Emerging Tech Roundup: ChatGPT Hype Fuels Urgency for Advancing Conversational AI and Generative AI

Emerging Tech: Venture Capital Growth Insights for Generative AI

Emerging Tech: Generative AI Needs Focus on Accuracy and Veracity to Ensure Widespread B2B Adoption

ChatGPT Research Highlights

Software Bill of Materials

Analysis By: Mark Driver

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

A software bill of materials (SBOM) is structured, machine-readable metadata that uniquely identifies a software package and the open source or proprietary components used to build it. SBOMs are designed to track and share the details of software components and their supply chain relationships across organizations. This enables greater transparency, auditability and traceability throughout the software supply chain, expediting resolution of security and compliance issues.

Why This Is Important

Gartner estimates that 40% to 80% of the lines of code in new software projects come from third parties (for example, runtime, libraries, components and software development kits [SDKs]). Most of this external code comes from myriad open-source projects; the remaining proprietary code comes from suppliers that provide little or no transparency to its status or condition. The quality and security of these external software assets (while leveraged across thousands of users) vary widely from one to another.

Business Impact

SBOMs aim to solve a fundamental problem with sharing open-source and third-party software. While organizations can use the same components, it's inefficient to duplicate efforts around tracking vulnerabilities and analyzing compliance violations. SBOM standards such as Software Package Data Exchange (SPDX) and CycloneDX establish a common infrastructure and a data exchange format to share details about software packages. This standardization reduces time to remediate issues through better collaboration between organizations.

Drivers

- **Software License Identification and Risk Assessment** — Like commercial software, open-source software (OSS) is almost always distributed according to the terms of a license agreement articulating the ways the software may be used. Some are considered permissive, others restrictive. Restrictive licenses could pose substantial legal risk and the loss of intellectual property. SBOMs can indicate the license used to distribute a particular software package in order to support the assessment of legal risks.

- **Need for Improved Security and Compliance** — Regulatory authorities in the U.S. (like the Food and Drug Administration [FDA] and Federal Energy Regulatory Commission [FERC]) and in Europe (the European Union Agency for Cybersecurity [ENISA]) are mandating SBOMs as a prerequisite deliverable for all organizations transacting with government agencies and regulated organizations. The goal of these regulations is to provide greater visibility into software components and dependencies to accurately determine an organization's exposure to security risks and vulnerabilities.

- **Need for Improved Visibility and Traceability of Software** — SBOMs can provide software engineering and vendor risk management teams with increased transparency into how software gets built, which components make up that software, and how quickly security vulnerabilities can be identified and remediated. When teams discover flaws or vulnerabilities in a component, they can use SBOMs to quickly identify all software that is affected by the vulnerable component. SBOMs also provide them with information to assess the potential impact and risks introduced by the vulnerable component.

Obstacles

- SBOMs are not a panacea. They're only as useful as the processes and tools implemented to process, analyze and leverage the information they contain. The impediments to adopting SBOMs include sharing SBOM data due to nonstandard data formats; inability to automate software delivery workflows based on the contents in the SBOM; and the ability to generate, consume and securely distribute them at speed and scale.

- The seamless integration of SBOMs into software development, packaging and release activities will be critical for their widespread adoption. The challenges in scaling the use of SBOMs include policy automation based on the information in the SBOM, secure distribution across organizational boundaries and managing their life cycle. The attestation of SBOMs for provenance is a prerequisite to treating them as a trusted source of dependency metadata. The dynamic nature of dependencies and their versions requires regenerating and updating SBOMs every time the artifacts are updated.

User Recommendations

- Enhance visibility into the complete software supply chain by tracking dependencies between open-source components in the software development life cycle (SDLC) using SBOMs.

- Discover affected components and share vulnerability data by using SBOM standards to exchange information about components and their relationships.

- Ensure SBOMs are rebuilt along with software artifacts by generating SBOMs during the software build process, rather than relying on pregenerated SBOM data. Use software composition analysis tools to generate and verify SBOMs for third-party, open-source components.

- Mitigate software supply chain security risks by requesting commercial software providers to provide standards-based SBOMs (for example, software package data exchange [SPDX], software identification [SWID], CycloneDX) during the procurement process.

Sample Vendors

Apiiro; Codenotary; Cybeats Technologies; Eracent; Finite State; Invicti Security (formerly NetSparker); OX Security; Revenera (Flexera); Rezilion

Gartner Recommended Reading

Emerging Tech: A Software Bill of Materials Is Critical to Software Supply Chain Management

Innovation Insight for SBOMs

Market Guide for Software Composition Analysis

How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks

Software Supply Chain Security

Analysis By: Dale Gardner

Benefit Rating: Transformational

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Software supply chain security (SSCS) is the set of processes and tools used to curate, create and consume software in ways that mitigate attacks against software or its use as an attack vector. Curation focuses on assessing risks of third-party software and assessing its acceptability. Creation focuses on secure development and the protection of software artifacts and the development pipeline. Consumption validates integrity of software through verification, provenance and traceability.

Why This Is Important

Given triple-digit increases in software supply chain attacks, global regulatory mandates, the widespread use of open-source tools and exposures due to remote ways of working, securing the overall software supply chain is paramount for all parts of an organization. This is particularly critical given growing digitized processes and services, making software the central location where most intellectual property now originates and resides.

Business Impact

- SSCS enables management and mitigation of a variety of risks, such as compliance failures and a host of security incidents.
- SSCS is necessary to satisfy both regulatory requirements and buyer demands for increased transparency around a vendor's application security measures.
- SSCS can help avoid increased friction and lost productivity from ad hoc efforts to secure the development environment and application artifacts.

Drivers

- Software supply chain attacks have become increasingly sophisticated, with malicious actors exploiting weaknesses at every stage in the software procurement, development and delivery life cycle. The number of attacks has increased dramatically, with the Sonatype report citing an average 742% year-over-year growth rate between 2019 and 2022. These attacks pose risks threatening the efficiency and productivity of business and organizations, and the basic ability of those organizations to operate. They may cause data breaches and other security incidents, and loss of intellectual property, and even pose threats to human safety.
- In response, governments and regulatory agencies throughout the world are turning their attention to ways in which organizations can be encouraged — or forced — to improve software supply chain security efforts. In the United States, a substantial focus is on the "power of the purse," with multiple initiatives across all sectors of the federal government focused on preventing vendors with poor security practices from reaching lucrative markets. These activities are not limited to the United States. Governments throughout North America, the European Union and the United Kingdom, and through Japan and Southeast Asia have taken actions ranging from the passage of legislation to efforts to improve awareness and understanding of the issues.
- Organizations have only just begun to tackle the issues associated with software supply chain risks, placing many in a "catch-up" position where action must be taken. They have been slow to take steps to secure their software supply chains. According to Sonatype's 8th Annual State of the Software Supply Chain Report, only about 7% of respondents have taken broad action. This is consistent with inquiry trends noted at Gartner. While there have been significant increases in inquiry around SSCS, many questions remain focused on specific subsets of the broader security challenge.

Obstacles

- The majority of organizations lack a complete understanding of SSCS, and thus have yet to adopt a comprehensive view of the software supply chain and all its risks. Such a view would encompass security through the curation and selection of secure software (during development and procurement), during creation, and in consumption as organizations track the usage of code and evolving threats. For example, many organizations are focused on acquiring software bills of material (SBOMs), but have yet to establish how those artifacts will be evaluated, stored, and used.
- Efforts to articulate secure means of ensuring the integrity and provenance of software artifacts throughout the software creation, curation, and consumption processes are emerging, but uneven in scope, execution and adoption. For example, it's common for organizations to evaluate open-source software used in development for vulnerabilities, but most organizations are not proactive when identifying potential risks.

User Recommendations

- Adopt a comprehensive view of software supply chain security and do not fall into the trap of focusing solely on specific aspects of the problem.
- When evaluating third-party software for use, proactively assess it for security, legal and intellectual property risks.
- Evaluate and authorize the use of specific components, and establish a trusted repository of vetted code. Track the usage and deployment of third-party code to ease remediation efforts when vulnerabilities are discovered.
- Protect the entire development tool chain and associated artifacts from attack. Include the protection of code and other artifacts (e.g., IaC or configuration scripts) from unauthorized access or alteration, defending the delivery pipeline from attack, and hardening the operating environment.

Sample Vendors

Arnica; BluBracket; Chainguard; Cybeats; Cycode; GitGuardian; Legit Security; Ox Security; Palo Alto Networks (Cider Security)

Gartner Recommended Reading

Emerging Best Practices to Manage Digital Supply Chain Risks

How Software Engineering Leaders Can Mitigate Software Supply Chain Security Risks

Innovation Insight for SBOMs

Performance Engineering

Analysis By: Joachim Herschmann

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Performance engineering is a systematic approach to developing software applications to ensure they meet the application performance objectives of the business. It focuses on the architecture, design and implementation choices that will affect application performance, and encompasses practices that help mitigate associated risks before progressing to subsequent phases.

Why This Is Important

The ability to consistently deliver products that satisfy end-user expectations of scalability, stability, quality of service (QoS), availability and performance has become crucial for digital businesses. Performance engineering promotes a holistic and proactive approach, with performance requirements driving the design, development and delivery of products as part of a continuous quality strategy.

Business Impact

Performance engineering includes both “shift left” and “shift right” testing practices and significantly improves an organization’s ability to consistently deliver solutions that exceed customers’ performance expectations. Organizations can improve application observability by using the insights gained through performance engineering. This includes adding metrics and telemetry, or deploying new application monitoring tools so alerts can be raised before a performance bottleneck impacts users.

Drivers

- Increased end-user expectations for application quality, specifically operational characteristics such as performance efficiency.
- The need to ensure business continuity under changing usage patterns, network topologies and data volumes.
- The need to optimize the use of modern microservices-based architectures, as well as the automation and integration capabilities of modern application platforms.
- Support for different migration scenarios, such as lift and shift, replatforming or refactoring the architecture of packaged or on-premises apps.
- The need to manage performance and scalability across different cloud providers, such as Amazon Web Services (AWS), Google or Microsoft Azure, to ensure the ability to shift from one operator to another without a change in user experience.
- Cost optimization for SaaS/PaaS services that makes use of dynamic infrastructure to spin up (and down) testing resources as needed.

Obstacles

- Many organizations focus only on tools and technology. Performance engineering requires a change in organizational culture. Tools enable quality but won’t solve problems on their own.
- Departmental silos, traditional top-down management structures and a lack of experience with managing quality continuously can impede adoption.
- Successful performance engineering requires clear goals that are aligned with the priorities of the business. The absence of established service-level indicators (SLIs) and objectives (SLOs) leads to a lack of realistic, quantifiable service availability or performance metrics.
- Performance engineering requires engaging stakeholders throughout the organization and empowering them to be more accountable and to seek out opportunities for improvement. Such a holistic approach can be seen as restrictive and requires consensus across all team members.
- Performance engineering includes designing with performance in mind, building the product with clear performance objectives and facilitating the discovery of issues early in development.

User Recommendations

- Create awareness of nonfunctional or operational characteristics, such as performance efficiency or the Application Performance Index (Apdex), an open standard developed by an alliance of companies to measure the performance of software applications in computing. ISO/IEC 25010 provides a template for understanding quality characteristics and includes performance efficiency as one of the top-level nonfunctional domains.
- Establish SLIs and SLOs as part of a performance engineering strategy. An SLI is a realistic, quantifiable measure of service availability or performance. An SLO is the target for the SLI over a fixed period.
- Foster a proactive performance quality strategy that makes performance an explicit requirement and verifies that performance goals are met and user satisfaction meets expectations.
- Allocate ownership and appoint staff with the skills needed for performance engineering by identifying the required roles, technologies and practices.
- Establish performance quality metrics based on the joint objectives that the business and IT are trying to accomplish.
- Collaborate with I&O leaders and establish a feedback loop using performance information from production and real users.

Sample Vendors

AppDynamics; Dynatrace; Keysight (Eggplant); Micro Focus; Perforce Software; Tricentis

Gartner Recommended Reading

How to Identify and Resolve Application Performance Problems

Quick Answer: Which Non-Functional Software Quality Characteristics Can Make or Break Your Product?

Improve Software Quality by Building Digital Immunity

AI-Augmented Testing

Analysis By: Joachim Herschmann, Jim Scheibmeir

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

AI-augmented testing comprises AI- and machine learning (ML)-based technologies and practices to make software testing activities more independent from human intervention. It continuously improves testing outcomes by learning from the data collected from performed activities. It extends traditional test automation beyond the automated execution of test cases to include fully automated planning, creation, maintenance and analysis of tests.

Why This Is Important

Software engineering leaders seeking to release faster without degrading quality are looking for more efficient ways of testing across all phases of the software life cycle. AI-augmented testing enables the automation of a broad set of testing activities related to requirement quality, design quality, code quality, release quality and operational resilience. This increases the degree of autonomy of those activities.

Business Impact

The adoption of AI-augmented testing has the potential to significantly improve an IT organization's ability to serve and delight its customers. It can enable the adjustment of testing scenarios and overall software quality parameters as part of a continuous quality strategy aimed at optimizing the end-user experience. It will also help to constitute a closed-loop system that provides continuous feedback about critical quality indicators and helps to reduce the costs of creating and maintaining tests.

Drivers

- A high dependency on human expertise and interaction limits how quickly modern digital businesses can design, build and test new software.
- Where automated testing is already in place, current levels of automation often remain below expectations due to a continued dependency on human intervention to maintain the automation as applications under test (AUT) evolve.
- The pressure to innovate quickly for market differentiation without compromising on quality relies on both a higher velocity and a higher degree of autonomy of the related activities.
- While delivery cycle time is decreasing, the technical complexity required to deliver a positive user experience and maintain a competitive edge is increasing. The answer is not more testing, but more intelligent testing enabled by AI technologies.

Obstacles

- Currently available tools are still relatively new, have a narrow scope and still need to prove their value. Generative AI, in particular, is the latest and most disruptive example. Hallucinations (content that is nonsensical or untruthful in relation to certain sources), subpar training data, potential copyright violations and security issues are the main risks associated with that particular set of AI technologies.
- Waiting until better AI-augmented testing solutions are available leads to a loss in competitive advantage and fewer innovations. It also incurs greater testing costs and the risk of undertesting.
- Underestimating the time required to acquire new skills and setting wrong expectations about the time required to become successful can be obstacles.
- Gathering, cleaning and processing data and training the model are not trivial tasks, and require adequate skills. Moreover, they are not yet autonomous processes.

User Recommendations

- Set the right expectations about the potential and limitations of AI-augmented testing and ensure that humans are always in the loop to verify the results produced by AI-augmented testing tools. This is particularly relevant for tools employing generative AI to automatically create tests, as generated tests may be completely useless or create false positives or negatives.
- Build a pilot team to map the cases where AI can provide the biggest benefits for software testing to the areas of greatest need in your organization. Evaluate opportunities to use it for test planning and prioritization, test creation and maintenance, test data generation, visual testing and test and defect analysis.
- Maximize the impact of AI-augmented testing by using it to enable a systematic approach to achieve the quality goals of business and development. Focus on key business value enablement and determine where it can help reduce cost and manage risk.

Sample Vendors

ACCELQ; Applitools; Avo Automation; CodiumAI; Diffblue; Functionize; mabl; ProdPerfect; testRigor

Gartner Recommended Reading

Market Guide for AI-Augmented Software-Testing Tools

Quick Answer: How Can AI Provide Benefits for Software Testing?

Innovation Insight for Continuous Quality

Improve Software Quality by Building Digital Immunity

Infographic: Artificial Intelligence Use-Case Prism for Software Development and Testing

Continuous Quality

Analysis By: Joachim Herschmann, Jim Scheibmeir

Benefit Rating: High

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Continuous quality is a systematic approach toward process improvement to achieve the quality goals of business and development. A continuous quality strategy fosters a companywide cultural change to drive quality ownership and engage everyone in achieving the set quality goals. It shifts focus from testing (an activity) to quality and digital immunity (outcomes) and encompasses the practices that help mitigate risks before progressing to subsequent stages of the software development life cycle.

Why This Is Important

The ability to consistently deliver business value with high quality has become critical for organizations seeking to mature their software development processes. Continuous quality encourages a holistic and proactive approach with functional and nonfunctional requirements driving the design, development and delivery of products. Development teams must use a continuous quality strategy to create quick feedback loops that enable easy adaptation to changes and increase customer satisfaction.

Business Impact

Continuous quality provides a framework for operational excellence that shifts the focus from testing as an activity to a holistic proactive approach to quality. Designing with quality in mind, building in quality and detecting defects earlier allow organizations to release higher-quality products more often than traditional quality control approaches enable. The adoption of a continuous quality strategy can significantly improve an organization's ability to serve and delight its customers.

Drivers

- Raised end-user expectations for application quality require a shift to a more holistic view of what constitutes superior quality that delights users.
- Organizations are under the pressure to innovate rapidly in order to launch differentiated products in the market quickly without compromising quality.
- Highly dynamic distributed cloud-based architectures require the ability to continuously monitor quality and deal with defects in real time.
- As software development scales, teams need consistent delivery practices and a clear understanding of the quality characteristics that make or break a product, especially about the nonfunctional (operational) aspects of the product.

Obstacles

- **Lack of clear goals:** Successful continuous quality requires clear goals aligned with the priorities of the business.
- **Internal pushback:** Continuous quality requires engaging stakeholders across the organization and empowering them to be more accountable. Such a holistic approach can be seen as too far-reaching and requires consensus on usage among all team members.
- **Loss of productivity:** Changing organizational culture and engaging in new practices require a significant amount of investment and time. This will impact current timelines and can cause a decrease in productivity prior to reaching steady productivity.
- **Limited to testing only:** Continuous quality includes designing a product with quality in mind, building it with clear quality objectives, and facilitating the discovery of issues early in the development process.

User Recommendations

- Move away from the traditional application- or project-centric- focused model to a holistic quality approach by adopting an ecosystem-centric view of quality and placing focus on business outcomes.
- Put a fundamental managed software testing approach in place with test policy and strategy being a focus area. Establish Test Maturity Model Integration (TMMi) Level 2 artifacts before assessing tools to understand opportunities for testing early and continuously.
- Accelerate product delivery by championing a continuous quality mindset and involving stakeholders across the organization. Allocate ownership and appoint staff with skills needed for continuous quality by identifying the required roles, technologies and practices.
- Enable collaboration with user experience (UX) designers and customer experience (CX) teams to infuse quality right from the inception of an idea. Establish relevant quality metrics based on the joint objectives that the business and IT are trying to accomplish.

Gartner Recommended Reading

Innovation Insight for Continuous Quality

Tool: Craft and Communicate a Continuous Quality Strategy

Infographic: Sketching a Continuous Quality Strategy Over Coffee

Improve Software Quality by Building Digital Immunity

Infographic: Essential Skills for QA Engineers

Observability

Analysis By: Padraig Byrne, Gregg Siegfried

Benefit Rating: Transformational

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Observability is the characteristic of software and systems that enables them to be understood, based on their outputs and enables questions about their behavior to be answered. Tools that facilitate software observability enable observers to collect and quickly explore high-cardinality telemetry using techniques that iteratively narrow the possible explanations for errant behavior.

Why This Is Important

The inherent complexity of modern applications and distributed systems and the rise of practices, such as DevOps, has left organizations frustrated with legacy monitoring tools and techniques. These can do no more than collect and display external signals, which results in monitoring that is, in effect, only reactive. Observability acts like the central nervous system of a digital enterprise. Observability tools enable a skilled observer to explain unexpected system behavior more effectively.

Business Impact

Observability tools have the potential to reduce both the number of service outages and their severity. Their use by organizations can improve the quality of software, because previously invisible (unknown) defects and anomalies can be identified and corrected. By enabling product owners to better understand how their products are used, observability supports the development of more accurate and usable software, and a reduction in the number and severity of events affecting service.

Drivers

- The term "observability" is now ubiquitous, with uses extending beyond the domain of IT operations. Although the 2020s are now the "decade of observability," care must be taken to ensure the term retains relevance when used beyond its original range of reference.
- OpenTelemetry's progress and continued acceptance as the "observability framework for cloud-native software" raises observability and its toolchain.
- Traditional monitoring systems capture and examine signals (possibly adaptive) in relative isolation, with alerts tied to threshold or rate-of-change violations that require prior awareness of possible issues and corresponding instrumentation. Given the complexity of modern applications, it is unfeasible to rely on traditional monitoring alone.
- Observability tools enable a skilled observer, a software developer or a site reliability engineer to explain unexpected system behavior more effectively, provided enough instrumentation is available. Integration of software observability with artificial intelligence for IT operations (AIOps) to automate subsequent determinations is a potential future development.
- Observability is an evolution of longstanding technologies and methods, and established monitoring vendors are starting to reflect observability ideas in their products. New companies are also creating offerings based on observability.

Obstacles

- In many large enterprises, the role of IT operations has been to "keep the lights on," despite constant change. This, combined with the longevity of existing monitoring tools, means that adoption of new technology is often slow.
- Enterprises have invested significant resources in their existing monitoring tools, which exhibit a high degree of "stickiness." This creates nontechnical, cultural barriers to adopting new practices such as those based on observability.
- Costs associated with observability tools have grown as companies struggle to keep up with the explosion in volume and velocity of telemetry.

User Recommendations

- Assess software observability tools to integrate into their continuous integration/continuous delivery (CI/CD) pipelines and feedback loops.
- Investigate problems that cannot be framed by traditional monitoring by using observability to add flexibility to incident investigations.
- Enable observability by selecting vendors that use open standards for collection, such as OpenTelemetry.
- Tie service-level objectives to desired business outcomes using specific metrics, and use observability tools to understand variations.
- Ensure IT operations and site reliability engineering teams are aware of updates to existing monitoring tools and how they may take advantage of them. Many traditional application performance monitoring vendors are starting to incorporate observability features into their products.
- Avoid the conclusion that observability is synonymous with monitoring. At minimum, observability represents the internal perspective, rather than external.

Sample Vendors

Chronosphere; Grafana; Honeycomb; Lightstep; Observe; VMware

Gartner Recommended Reading

Monitoring and Observability for Modern Infrastructure and Applications

Magic Quadrant for Application Performance Monitoring and Observability

DevOps Test Data Management

Analysis By: Andrew Bales

Benefit Rating: Moderate

Market Penetration: 20% to 50% of target audience

Maturity: Early mainstream

Definition:

DevOps test data management is the process of providing DevOps teams with sanitized data to evaluate the performance, functionality and security of applications. It typically includes copying production data, anonymization or masking, and sometimes, virtualization. In some cases, specialized techniques, such as synthetic data generation, are appropriate. Given potential compliance and privacy issues, the efforts frequently involve members of application and data security teams.

Why This Is Important

Test data management is inconsistently adopted across organizations, with many teams still copying production data for use in test environments. As organizations shift to DevOps and the pace of development increases, this traditional approach is increasingly at odds with requirements for efficiency, privacy and security, and even the increased complexity of modern applications. This opens organizations to a variety of legal, security and operational risks.

Business Impact

Quick provisioning of test data helps ensure the pace of development isn’t slowed. It’s also increasingly important to remain compliant with the growing number of privacy mandates to which organizations are subject. This helps avoid fines and remediation and mitigation costs, along with the inevitable delays associated with audits and investigations. Finally, by providing application teams with anonymized or synthetic data, the risk of data breaches is reduced.

Drivers

- Test data management is generally viewed as a mature, relatively uncomplicated, practice. However, the reality is the combination of the increased pace of development from DevOps and a growing number of privacy mandates and constraints have stressed traditional approaches — prompting the use of virtualization as well as alternative masking and protection techniques, including synthetic data generation.
- More traditional test data management has been inconsistently adopted, with many organizations either simply using copies of production data in unsafe environments or generating “dummy data” (distinct from emerging synthetic data generation techniques) that doesn’t accurately reflect production data. The data privacy requirements and complexity issues noted have prompted organizations to revisit and update their processes with an eye toward scalability and automation. Updated technologies may also be a requirement. For example, requirements for speed and agility have created a need for data virtualization tools.
- Data protection is cited by most Gartner clients in inquiries regarding test data management. Privacy and data protection requirements mean it’s no longer safe to simply provide development teams with a copy of production data since this practice leaves organizations open to increased risk of regulatory violations, data breaches and other security issues.
- With modern applications relying on an increasing number of interconnected data stores (many of which, technologically speaking, are vastly more complex) and applications and APIs to function, testing has become more complex. Such complexity demands that tools support the ability to coordinate and synchronize changes across different data stores to ensure relational consistency while still addressing security and speed mandates.

Obstacles

- In the absence of a strong culture of security, processes and technologies to protect sensitive information during development and testing will encounter friction. Conflicting needs for rapid development and privacy require attention to a mix of organizational and cultural issues to strike a balance across groups.
- Responsibility for test data management in organizations has been shared by application development and database administration. New technologies and processes may shift those responsibilities to include security, complicating organizational dynamics and potentially creating the need for additional resource allocation.
- Implementation can be a burden, especially where little or no data sensitivity classification has been done. This must be accomplished before teams can proceed with the required data transformation and masking. These efforts are typically combined with an analysis of data relationships so that relational integrity can be assured.

User Recommendations

- Involve stakeholders — such as data management, privacy and security teams, compliance teams, etc. — to understand consumption patterns and needs — as appropriate.
- Document existing test data management practices so tools and processes can be evaluated against data protection mandates.
- Coordinate with other teams to avoid duplication of effort and tooling since data masking tools may also be used by analytics teams (e.g., to provide data for machine learning or other purposes).
- Evaluate data masking tooling by considering support for databases and other stores, data discovery capabilities, types of masking supported, and the ability to coordinate change to ensure consistency (e.g., key fields) across multiple sources.
- Evaluate data virtualization for DevOps use cases where frequent updates to test data are required. Virtualization can speed up the process of providing copies of safe data.
- Determine whether synthetic data generation is appropriate in cases where suitable data doesn’t exist or reidentification risks are high.

Sample Vendors

BMC Software; Broadcom; DATPROF Delphix; Hazy; Informatica; K2View; Mage; OpenText; Solix Technologies

Gartner Recommended Reading

Market Guide for Data Masking

Elevating Test Data Management for DevOps

3 Steps to Improve Test Data Management for Software Engineering

Innovation Insight for Synthetic Data

Sliding into the Trough

Site Reliability Engineering

Analysis By: George Spafford, Daniel Betts

Benefit Rating: Transformational

Market Penetration: 5% to 20% of target audience

Maturity: Adolescent

Definition:

Site reliability engineering (SRE) is a collection of systems and software engineering principles used to design and operate scalable resilient systems. Site reliability engineers work with the customer or product owner to understand operational requirements and define service-level objectives (SLOs). Site reliability engineers work with product or platform teams to design and continuously improve systems that meet defined SLOs.

Why This Is Important

SRE emphasizes the engineering disciplines that lead to resilience; but individual organizations implement SRE in widely varying ways such as a defined role or a set of practices. SRE teams can serve as an operations function, and nearly all such teams have a strong emphasis on blameless root cause analysis. This is to decrease the probability and/or impact of future events and to enable organizational learning, continual improvement and reductions in unplanned work.

Business Impact

The SRE approach to improving reliability and resilience is intended for products and platforms that need to deliver customer value at speed at scale while managing risk. The two primary use cases are to improve the reliability of existing products/platforms or to create new products or platforms that need reliability from the start.

Drivers

- Clients are under pressure to meet customer requirements for reliability while scaling their digital services and are looking for guidance to help them.

- While Google originated what became known as SRE and continued to evolve it, practitioners are developing and sharing new practices as well. Potential practitioners looking for pragmatic guidance to improve the reliability of their systems have a rich body of knowledge they can leverage that works well with agile and DevOps.
- Organizations are adopting highly skilled automation practices (usually DevOps), and usage of infrastructure-as-code capabilities (which usually requires a cloud platform) to deliver digital business products reliably.
- The most common use case based on inquiry calls with clients is to leverage SRE concepts to improve the reliability of existing systems that are not meeting customer requirements for availability, performance or are proving difficult to scale.

Obstacles

- Insufficient internal marketing to understand what agile, DevOps or product teams need or would value and then explaining how the value SRE can deliver will justify the costs and risks incurred. Without marketing its benefits, SRE adoption tends to be less certain or slower. The SRE concept by itself is insufficient — people must continuously believe it is worthwhile.
- Finding SRE candidates who have the right mix of development, operations and people skills is a big challenge for clients. Impacts on initial adoption and scaling efforts as well.
- Rebranding of a traditional operations team without changing to adopt SRE practices, only SRE in name.
- Clients have voiced problems with product owners who overly focus on functional requirements and not nonfunctional requirements thus slowing improvements and support of SRE within the organization.

User Recommendations

- Leverage practices pragmatically based on need. Don’t feel that you must implement SRE exactly the way Google does it, learn what works for you.
- Detect an opportunity to begin that is politically friendly, will demonstrate sufficient value and has an acceptable risk profile.
- Start small, focus, learn, improve, and demonstrate value — do not try to change everything at once.

- Work with the customer or product owner to define clear, obtainable SLOs based on their needs.
- Implement monitoring and improve observability to objectively report on actual performance relative to the SLOs.
- Product owners must be accountable for functional and non-functional requirements of their products.
- Instill collaborative working between site reliability engineers, developers and other stakeholders to help them learn how to design, build and evolve their products to meet SLOs.
- Create a community, implement effective organizational learning practices and evolve SRE practices.

Sample Vendors

Atlassian; Blameless; Datadog; Dynatrace; New Relic; OpsRamp; PagerDuty; Splunk

API Security Testing

Analysis By: Dale Gardner

Benefit Rating: High

Market Penetration: 20% to 50% of target audience

Maturity: Adolescent

Definition:

API security testing is a specialized type of application security testing (AST) aimed at identifying vulnerabilities in APIs. Checks should include both traditional application vulnerabilities (such as injection attacks) and API-specific issues (such as broken-object-level authorization). Availability of an API specification (such as OpenAPI) is sometimes a prerequisite for effective testing. Discovery technologies help ensure that unknown APIs are identified and tested for vulnerabilities.

Why This Is Important

APIs represent a major attack surface for web-enabled applications. Attacks on and abuse of APIs result in serious adverse consequences, including data breaches and other security incidents. DevSecOps teams focus on the need for API security testing in development to prevent this, but APIs pose unique risks. Many organizations rely on traditional AST tools extended to support these requirements or speciality tools designed for APIs. Some vulnerabilities may require manual testing to detect.

Business Impact

APIs are a foundational element of many organizations' digital transformation efforts. Hence, securing APIs from attack and misuse is a continuing concern for many security and risk management (SRM) professionals. API-specific testing — pre- and postdeployment — builds a solid foundation for an overall API security strategy. Automated API discovery is needed because many organizations struggle to maintain an inventory of APIs and need help locating them so they are tested and managed.

Drivers

- In support of digital transformation efforts, APIs have become common in application architectures to enable information flow and support transactions between processes, applications and systems. However, that growth has brought increased attention from attackers, and APIs have become the primary attack surface for many systems.

- API attacks have resulted in a stream of data breaches and other security incidents, yielding significant damage to organizations and individuals. As a consequence, DevSecOps teams — along with the business leaders whose applications are supported by APIs — are increasingly interested in API testing and security.

- Because the creation, development and deployment of APIs may be loosely managed, security teams often have to contend with undocumented or shadow APIs, which exist outside normal processes and controls. Such APIs may be especially deficient in secure design and testing, posing an even greater risk. Hence, the discovery of APIs deployed to production is another important need.

- API security testing helps avert the tangible and intangible costs associated with breaches and other types of security incidents.

- Traditional AST tools — SAST, DAST and interactive AST (IAST) — were not originally designed to test for some of the unique types of vulnerabilities associated with typical attacks against APIs, or for newer types of APIs (such as GraphQL). API-specific vulnerabilities and modern API formats prompt security and development teams to implement specialized API security tools that are focused on testing, discovery of shadow APIs and protection from threats (or some combination of the three).

Obstacles

- Traditional tools offer inconsistent support for detecting API-specific vulnerabilities. APIs are susceptible to most traditional application attacks, but other attacks are common. For example, improper authorization checks around object identifiers have been cited in a number of breaches. Specialized tools or penetration testing may be needed for reliable detection of these flaws.

- Testing tools may require an API specification to perform effective testing.

- Testing tools may not support all API protocols. SOAP remains in widespread use, although it is being supplanted by REST APIs. GraphQL-based APIs are increasingly common, so additional support is required from tool vendors for effective testing.

- Some confusion remains in the marketplace, with various types of companies offering products (including traditional AST vendors and API testing and protection vendors), complicating evaluation and selection efforts.

User Recommendations

- Begin evaluation and selection efforts by focusing on the criticality of APIs; their security and business risks; and the technical requirements they pose.

- Examine the testing and discovery capabilities provided by existing tools in your application security portfolio. Many have added support for APIs, although actual tests may still focus primarily on traditional application vulnerabilities and lack support for API-specific vulnerabilities.

- Evaluate newer alternatives where gaps are found. They may also offer additional options, including audit of design-stage API specifications, discovery, vulnerability scans and threat protection.

- Complement automated testing tools with penetration testing services for comprehensive coverage as traditional AST is often unable to detect some common API-specific vulnerabilities.

- Evaluate the ability of a given tool to address multiple requirements. API security tool capabilities — including discovery, testing and threat protection — often overlap.

Sample Vendors

42Crunch; APIsec; Contrast Security; HCLSoftware; Noname Security; OpenText; PortSwigger; StackHawk; Synopsys; Veracode

Gartner Recommended Reading

API Security: What You Need to Do to Protect Your APIs

API Security Maturity Model

How to Deploy and Perform Application Security Testing

Research Index: Everything You Should Do to Address API Security

Critical Capabilities for Application Security Testing

Product-Centric Delivery Model

Analysis By: Mike West, Sarah Davies

Benefit Rating: High

Market Penetration: More than 50% of target audience

Maturity: Mature mainstream

Definition:

A product-centric delivery model allows you to take advantage of product management. That is to structure your organization's operating model to create small teams focused on product development and provide greater flexibility to meet the shifting demands of consumers. This allows organizations to adapt how software is consumed, and enables software leaders to shift from time-bound episodic delivery to continuous delivery.

Why This Is Important

The ability to respond to market trends, the economy and shifting consumer demands without complex organizational changes is driving organizations to adopt the product-centric delivery model. This model allows organizations to have greater control of their enterprise strategy and move away from the standard, functional skill-based organization and embrace multiskilled persistent teams that work together on a product or product line.

Business Impact

A product-centric delivery model enables an enterprise to:

- Focus on outcomes rather than functions, and incremental improvements to measured business outcomes.
- Use venture capital or investment funding models as financial methods to control investment at operational levels.
- Improve agility to respond to changing market demands and customer value prioritization.
- Reduce silos, improve collaboration across product value streams, and have flatter organization and more rapid decision making.

Drivers

- Organizations must adjust their delivery models to keep pace with market demands and increased volatility.
- Investment and financial models need to provide flexibility and support evidence-based market research and responses to corporate strategy.
- Organizations need rapid, incremental feedback that engineering teams can respond to flexibly to satisfy and delight customers.
- A shift to cloud-based architecture is driving the adoption of value-based operating models that reflect the customer journey rather than existing management frameworks.
- The continuous disbanding of project teams has left today's organizations feeling the need to address new talent retention strategies and to overcome inefficiencies caused by siloed data and solutions via continuous innovation and delivery.

Obstacles

The key factors hampering the adoption of product-centric delivery models include:

- Inertia from existing organizational culture and management frameworks reluctant to disband current budgets and authority positions. This is compounded by the difficulty finding experienced product management subject matter experts, e.g., product managers, to help overcome this reluctancy.

- Walls between business and IT due to the lack of alignment around outcomes, responsibilities, siloed budgets and success metrics, which leads to a lack of understanding of business outcome metrics such as leading key performance indicators (KPIs) and objective and key results (OKRs).
- The lack of senior management and organizational support, which leaves adoption in pockets across the organization, and outmoded governance processes incentivizing control and risk aversion rather than experimentation and innovation.

User Recommendations

- Establish clear goals and objectives for the transition, anchored on business priorities, building leadership support for the necessary culture and governance change.
- Establish a strong partnership with colleagues as you identify and train product managers, product owners, business leaders and team members on agile and product management practices.
- Transform governance to embrace business architecture practices such as value stream mapping, business capability modeling, and customer and employee journey mapping.
- Move to a product funding and work prioritization model that allows for reallocation of resources based on business demand and changing market conditions. Create an explicit network of dashboards to convey the outcomes of product initiatives. Manage recurring reviews of outcomes to assess the value of work underway.

Gartner Recommended Reading

Strengthen Five Key Pillars of Product Management to Scale for Digital Business Success

Prepare Now for the Future of Digital Product Management

Improve Product Team Speed and Agility by Minimizing Dependencies: Approaches From 3 Leading Organizations

Overcome Objections and Sell the Benefits of Moving From Projects to Products and Agile

Create a Product Operations Role to Improve the Strategic Focus of Product Managers

Behavior-Driven Development

Analysis By: Jim Scheibmeir, Joachim Herschmann

Benefit Rating: Moderate

Market Penetration: 20% to 50% of target audience

Maturity: Early mainstream

Definition:

Behavior-driven development (BDD) expresses requirements as customer behaviors linked to scenarios and outcomes to enhance collaboration between business domain experts, product owners, developers and testers. BDD uses domain-specific language and is implemented with executable specifications from feature files. BDD grew out of test-driven development (TDD) and acceptance-test-driven development — practices that shift software testing left in software engineering activities.

Why This Is Important

BDD extends the concept of TDD closer to the business user by providing a mechanism to declare functionality expectations in terms of behaviors. BDD improves traceability and auditability by providing a clear trail from the feature to the code that implements it — and vice versa. BDD supports collaboration between business experts, testers and developers.

Business Impact

Product development teams working in agile or DevOps should use BDD to improve collaboration on software products considering the application behaviors and user experience (UX). Because BDD scripts are accessible to a nontechnical audience, they can be developed with business technologists, helping the software fulfill the acceptance criteria. Writing the tests before coding clarifies requirements and maximizes the benefits of automated tests; it ensures tests are available as early as possible.

Drivers

- The market demands outstanding UX and fast time to market, such as organizational and business dependencies on digital, heightened expectations from users.
- BDD offers a path to agile adoption for organizations in highly regulated industries requiring detailed documentation or traceability.
- BDD facilitates the use of specifications to drive acceptance test automation.

Obstacles

BDD's adoption impediments are related to process changes, involvement from business and technical roles, and the skill sets it requires. Software engineering leaders struggle to ensure the adoption of BDD because they fail to adequately influence and manage:

- **Generative AI and natural language processing (NLP):** New generative and NLP capabilities in products will disrupt BDD's collaborative practice adoption.
- **Internal pushback:** BDD requires the engagement of business and IT stakeholders and empowers them to be more accountable.
- **Temporary loss of productivity:** Changing organizational culture and engaging in new practices require significant investments of money and time.
- **Limited value and ROI:** BDD alone does not guarantee the maintainability or security of written code, nor the performance of a solution.
- **Learning curve:** The Given-When-Then expression of Gherkin, a common domain-specific language (DSL), does not fit how every organization expresses software requirements.

User Recommendations

- Start BDD at the beginning of the product life cycle to improve requirements and acceptance criteria. Focus on quality in user stories through collaboration between technical and nontechnical staff. Specify application conditions and behaviors within feature files.
- Give teams support and time in creating guidelines for the Gherkin implementation and for building expertise.
- Improve the development process and specifically the areas around software quality by involving stakeholders, developers and testers in creating scenarios and feature files.
- Describe the desired functionality using the Gherkin Given-When-Then syntax before writing the code. Use BDD tools to build automated tests that are verifiable by the product owner and business domain experts.
- Invest in training to expand your agile teams' understanding of test automation skills, and customer and application behaviors.

Sample Vendors

froglogic; Katalon; SmartBear Software; Tricentis

Gartner Recommended Reading

Predicts 2023: Observing and Optimizing the Adaptive Organization

Innovation Insight for Continuous Quality

Essential Skills for Agile Development

Predicts 2023: How Innovation Will Transform the Software Engineering Life Cycle

Top Strategic Technology Trends for 2023: Digital Immune System

Climbing the Slope

Feature Management

Analysis By: Keith Mann

Benefit Rating: High

Market Penetration: 20% to 50% of target audience

Maturity: Early mainstream

Definition:

A feature is a discrete unit of software that provides a single, valuable function or capability within a larger system. Feature management combines the selective enabling or disabling of features via feature toggles (also known as feature flags) with the monitoring, assessment, and comparison of features and feature variants.

Why This Is Important

Feature management enables the use of feature toggles on a very large scale without incurring technical debt, which in turn enables them to be used for more purposes. Although feature toggles were originally used to disable features that were not ready for deployment, they are now also used for experimentation, testing and progressive release. Features are increasingly being monitored and tested for value throughout their life cycle, creating feedback loops that progress feature design.

Business Impact

Almost all businesses can benefit from faster, more reliable delivery of software — the impact of feature management in that sense could be universal. Experimentation has already had a large impact on digital marketing and digital consumer product development, where human responses are hard to predict. The biggest potential impact comes from the improved ability to select valuable features for development, based on feedback loops from feature monitoring in production to development planning.

Drivers

- Software engineering teams are reaching high levels of maturity in their use of agile and DevOps practices. This enables them to deliver software even faster than it can be deployed and consumed. Software engineering organizations can use feature management to decouple delivery from deployment and avoid slowing down teams.

- Demand for continuously available systems is increasing. This drives a need for instant recovery from defective features, which in turn fuels a need for feature toggles and feature management.
- With new technologies and corresponding new user experience options comes uncertainty about which types of experience are most effective for users. Feature management enables experiments that reduce this uncertainty by presenting various experiences to various sample audiences and measuring their responses.
- Advanced software engineering organizations are finding that fast delivery alone does not satisfy stakeholders. They are beginning to realize that they must establish feedback loops from production to feature design so that they can learn which designs produce the greatest value. Feature management enables the value of features to be monitored, tracked and compared.
- Speed to market pressures can be met in part through progressive releases — which enable features to be rolled out as their capabilities, such as localization, expand to meet the needs of new user segments.

Obstacles

- Even years after its genesis, the feature management market is still dominated by fairly small vendors. This limits user awareness and adoption.
- Feature management vendors are just beginning a shift from a technical to a business focus in their messaging. Technology buyers, like software engineering leaders, are often responsible for justifying the investment.
- The vision of feature management as a tool for experimentation and value feedback is still new. The focus has long been on the speed and reliability of deployment, so buyers view that as the benefit of feature management. Shifting the message is proving to be hard.
- Although value feedback loops could be the most important application of feature management, the concept and corresponding practices are still emerging.
- As with many areas of software engineering, low talent availability is a problem. Data scientists and other professionals skilled in experimentation are in short supply and high demand.

User Recommendations

- Ensure that development teams are familiar with feature management techniques. Ideally, developers will already be using feature toggles, where appropriate, to improve deployment speed and reliability.
- Promote the use of feature management for experimentation. This may mean selecting a feature management vendor and training or hiring staff.
- Introduce the concept of value feedback loops to software designers. Establish value feedback loops as tools and practices mature.
- Have software designers clearly identify features during solution design, and define the value of each feature.
- Adopt progressive release practices to reduce change impacts and enable earlier delivery.

Sample Vendors

Amplitude; Kameleoon; LaunchDarkly; Optimizely; Split; Unleash

Gartner Recommended Reading

Software Engineering Teams Must Learn to Deliver More Value

Data-Driven DevOps: How to Rethink the Measurement Approach

Adopt Platform Engineering to Improve the Developer Experience

Secure Coding Training

Analysis By: Aaron Lord, Manjunath Bhat

Benefit Rating: High

Market Penetration: 20% to 50% of target audience

Maturity: Adolescent

Definition:

Secure coding training raises awareness of the impact and prevention of vulnerabilities in source code. Developers are trained in secure coding practices for specific coding languages and frameworks using different methods like just-in-time (JIT) training, gamified lessons, videos, workshops and challenges.

Why This Is Important

Seventy-five percent of software engineering leaders surveyed in the 2022 Gartner Software Engineering Leaders Role Survey stated that application security skills are a pain point in their organization. Secure coding skills evolve over time and are lacking from general software engineering education. Ensuring that software engineers are up-to-date with the latest secure coding skills will raise awareness of the risk of introducing vulnerabilities.

Business Impact

Any organization that writes software needs to be concerned about security and reliability. Software engineering leaders and practitioners need to implement secure coding training with regular cadence. Secure coding training takes multiple approaches to learning, each with their own pros and cons. Secure coding training vendors are leaning heavily on the workshop and gamified approach, although that will be less effective for more established organizations with seasoned developers.

Drivers

The creation of secure software requires software engineers have the skills to apply secure coding practices:

- Application security is a growing concern for organizations that create software.
- Developers lacking security understanding produce unsecure applications by introducing vulnerable code, third-party components and infrastructure configurations.
- Staying up-to-date with secure coding practices will help reduce the risk of data loss and breaches.
- Organizations that implement security champions or coaches need to enable advanced learning for these individuals.

Obstacles

Enabling learning for software engineers, especially at scale, has a number of challenges:

- Just like any approach to teaching, types of secure coding training work better or worse for certain individuals (presentation, tests, workshops, etc.).
- Software engineer leaders are not investing in secure coding training and offering no incentives for its completion.

- When training is assigned annually, dropping knowledge retention between training sessions may lead to security mistakes.
- Offerings and pricing models for secure coding training can vary wildly, so it can be difficult to understand what is needed.
- Secure coding training has yet to be effective at a large scale across multiple regions.

User Recommendations

Take into account multiple factors for your engineering organization that apply to secure coding training needs to:

- Utilize a security champions program to push training and security awareness from within.
- Ensure that a secure coding training vendor offers training that matches the organization’s technology stacks, languages and frameworks.
- It is preferable to use training that has a mix of approaches instead of just one (presentations, workshops, challenges, gamification, etc.).
- Ensure secure coding training offerings come in multiple languages that can support engineers across multiple regions.
- Integrate secure coding training into software development life cycle (SDLC) tooling for JIT training to maintain knowledge retention over time.

Sample Vendors

Avatao; Codebashing; Immersive Labs; Secure Code Warrior; SecureFlag; Security Compass; Security Journey; Snyk; Synopsys; Veracode

Gartner Recommended Reading

How to Select DevSecOps Tools for Secure Software Delivery

Infographic: Build These Essential Skills to Secure Modern Software Development

Develop Your Technical Skills Using Online Learning Platforms

Entering the Plateau

Citizen Developers

Analysis By: Jason Wong

Benefit Rating: High

Market Penetration: More than 50% of target audience

Maturity: Mature mainstream

Definition:

Citizen developers are employees not in the IT organization who create or extend technology capabilities. They use low-code, no-code and generative AI development tools and runtime environments sanctioned by corporate IT or the business units. A citizen developer is a subsegment of business technologists. However, it is not a title, role or professional developer in the business unit, but rather a persona taken on by an employee.

Why This Is Important

Defining and embracing citizen development is essential to the maturing digital workplace strategy. The changing ways of work have accelerated the need for greater business agility and putting better tools in the hands of employees so they can more rapidly solve their problems with new digital capabilities. According to the 2022 Gartner Digital Worker Survey, 48% of non-IT workers customize or build tools from technology provided by IT or from tools they acquire on their own (see What Workers Want: Top 10 Insights From the Digital Worker Experience Survey).

Business Impact

The long-term strategic impact of citizen development is enabling self-service business innovation within business units and fostering fusion teams that blend business and technology expertise. Citizen developers are often aided by IT in some aspects of co-creation or technical support. Citizen development communities of practice and hackathons have proven to help promote and enhance digital dexterity across the enterprise.

Drivers

- The workforce is changing to being more tech savvy. On average, 69% of the respondents in the 2022 Gartner Digital Worker Survey stated improving their personal digital skills to be important for business success.

- Employees have easier access to more tools than ever before, and it's only increasing. Citizen developers feel more empowered by powerful low-code development tools and SaaS-based no-code tools that specifically cater to them. Many business application vendors now provide robust low-code and no-code development capabilities, making it easier for citizen developers to develop their own solutions — even ones that once required professional development skills, such as building mobile apps or using AI automation services like chatbots.

- The nature of work involves using more technologies. Citizen developers may also take on other citizen technologist personas depending on their skills, ambition and scope of work. Gartner often sees citizen data scientist, citizen integrator and citizen automator personas in the digital workplace. Over time, some of these citizen developers have become part of fusion teams that drive business and IT collaboration and development.

Obstacles

- Citizen development is not shadow IT. IT's resistance to recognizing business technologists' work and embracing citizen development results in missed opportunities to drive toward business and IT alignment.

- IT leaders also often fear losing control on account of increasing citizen development activities, making their teams less relevant or burdening IT with unmaintainable apps. However, the risks of citizen development are typically outweighed by the benefits. Risks to the enterprise can be better managed by directly addressing inadequate tooling and disorganized support for a citizen development community, which are key factors leading to poor outcomes and risky apps.

- IT leaders often don't understand the levels of ambition that exist in their organization and don't have a plan to support those ambitions. Citizen development is on a digital dexterity continuum that progresses from digital citizen to digital side hustle to business technologist.

User Recommendations

- Engage tech-savvy business users more actively to enlist and enable them to become citizen developers. Ignoring or attempting to prevent citizen development often carries more risks and limits enterprise innovation.

- Mitigate risks by working with business unit leaders and citizen developers to establish trust; clarify ownership and accountability expectations; and define safe activity zones.

- Enable self-governing citizen development practices by fostering a community of practice (CoP) across business units and with IT.
- Improve outcomes for citizen-developed apps by joint (business and IT) selection of the right tools and enabling technologies.

Sample Vendors

Airtable; Creatio; Kissflow; Microsoft; Project Management Institute (PMI); Quixy

Gartner Recommended Reading

Quick Answer: What Types of Fusion Teams Do Business Technologists Lead?

Quick Answer: How Can Digital Workplace Leaders Support Business Technologists?

What Are the Digital Dexterity Skills Necessary to Support New Ways of Working?

Case Study: Kick-Starting a Low-Code/No-Code Community of Practice (Heathrow Airport)

DevSecOps

Analysis By: Neil MacDonald, Mark Horvath

Benefit Rating: Transformational

Market Penetration: More than 50% of target audience

Maturity: Mature mainstream

Definition:

DevSecOps is the integration and automation of security and compliance testing into agile IT and DevOps development pipelines, as seamlessly and transparently as possible, without reducing the agility or speed of developers or requiring them to leave their development toolchain. Ideally, offerings provide security visibility and protection at runtime as well.

Why This Is Important

DevSecOps offers a means of effectively integrating security into the development process, in a way that eliminates or reduces friction between security and development. The goal is to pragmatically achieve a secure, workable software development life cycle (SDLC) supporting rapid development. DevSecOps has become a mainstream development practice, although the specifics can vary between organizations based on their technology and the maturity of their development processes.

Business Impact

The goal of DevSecOps is to speed up development without compromising on security and compliance. Furthermore, the externalization of security policy enables business units and security organizations to define and prioritize policy guardrails and lets developers focus on application functionalities. Policy-driven automation of security infrastructure improves compliance, the quality of security enforcement and developer efficiency, as well as overall IT effectiveness.

Drivers

- Adoption of DevOps, and other rapid development practices, requires security and compliance testing that can keep up with the rapid pace of development.
- DevSecOps offerings are applied as early as possible in the development process, whereas traditional application security testing (AST) tools associated with older development models are applied late in the development cycle, frustrating developers and business stakeholders.
- Testing results need to be integrated into the development process in ways that complement developers' existing workflows and toolsets, and not require them to learn skills unrelated to their goals.
- The use of open source has greatly increased the risk of the inadvertent use of known vulnerable components and frameworks by developers.

Obstacles

- Incorrectly implemented, siloed and cumbersome security testing is the antithesis of DevOps. Due to this, developers believe security testing tools are slowing them down.
- Developers don't understand the vulnerabilities their coding introduces.
- Developers don't want to leave their development (continuous integration/continuous delivery [CI/CD]) pipeline to perform tests or to view the results of security and compliance testing tools.
- Historically, static application security testing (SAST) and dynamic application security testing (DAST) tools have been plagued with false positives or vague information, hence frustrating developers.
- The diversity of developer tools used in a modern CI/CD pipeline will complicate the seamless integration of DevSecOps offerings.

User Recommendations

- "Shift left" and make security testing tools and processes available earlier in the development process.
- Prioritize the identification of open-source software (OSS) components and vulnerabilities in development (referred to as software composition analysis).
- Opt for automated tools with fast turnaround times, with a goal of reducing false positives and focusing developers on the highest-confidence and most-critical vulnerabilities first.
- Ask vendors to support out-of-the-box integration with common development tools and support full API enablement of their offerings for automation.
- Evaluate emerging cloud native application protection platform (CNAPP) offerings for technical control implementation.
- Require security controls to understand and apply security policies in container- and Kubernetes-based environments.
- Favor offerings that can link scanning in development to correct configuration, visibility and protection at runtime.

Sample Vendors

Apiiro; Aqua Security; Contrast Security; Dazz; Lacework; Palo Alto Networks; Qwiet AI; Snyk; Sonatype; Wiz

Gartner Recommended Reading

How to Select DevSecOps Tools for Secure Software Delivery

Market Guide for Cloud-Native Application Protection Platforms

Magic Quadrant for Application Security Testing

12 Things to Get Right for Successful DevSecOps

How to Manage Open-Source Software Risks Using Software Composition Analysis

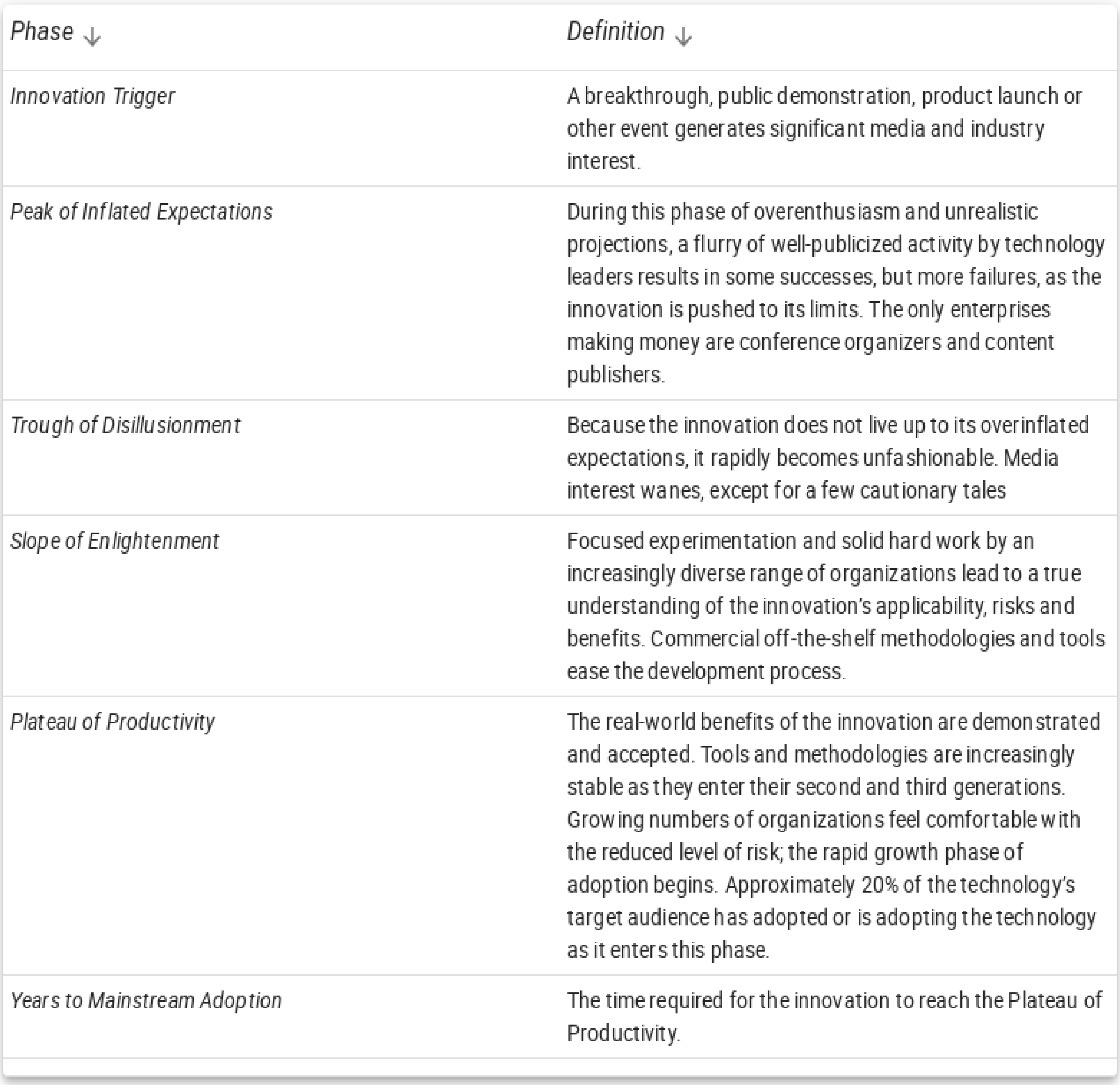
Appendixes

See the previous Hype Cycle: Hype Cycle for Software Engineering, 2022.

Hype Cycle Phases, Benefit Ratings and Maturity Levels

Table 2: Hype Cycle Phases

(Enlarged table in Appendix)



Phase ↓	*Definition* ↓
Innovation Trigger	A breakthrough, public demonstration, product launch or other event generates significant media and industry interest.
Peak of Inflated Expectations	During this phase of overenthusiasm and unrealistic projections, a flurry of well-publicized activity by technology leaders results in some successes, but more failures, as the innovation is pushed to its limits. The only enterprises making money are conference organizers and content publishers.
Trough of Disillusionment	Because the innovation does not live up to its overinflated expectations, it rapidly becomes unfashionable. Media interest wanes, except for a few cautionary tales
Slope of Enlightenment	Focused experimentation and solid hard work by an increasingly diverse range of organizations lead to a true understanding of the innovation's applicability, risks and benefits. Commercial off-the-shelf methodologies and tools ease the development process.
Plateau of Productivity	The real-world benefits of the innovation are demonstrated and accepted. Tools and methodologies are increasingly stable as they enter their second and third generations. Growing numbers of organizations feel comfortable with the reduced level of risk; the rapid growth phase of adoption begins. Approximately 20% of the technology's target audience has adopted or is adopting the technology as it enters this phase.
Years to Mainstream Adoption	The time required for the innovation to reach the Plateau of Productivity.

Source: Gartner (August 2023)

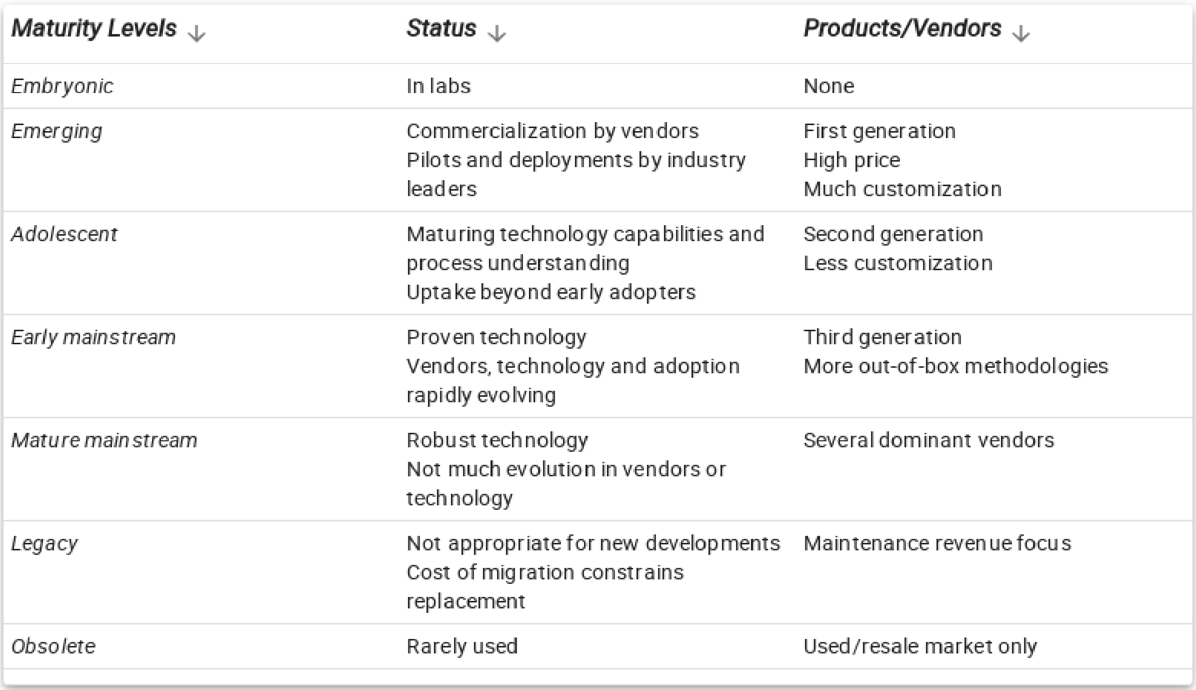
Table 3: Benefit Ratings

Benefit Rating ↓	***Definition*** ↓
Transformational	Enables new ways of doing business across industries that will result in major shifts in industry dynamics
High	Enables new ways of performing horizontal or vertical processes that will result in significantly increased revenue or cost savings for an enterprise
Moderate	Provides incremental improvements to established processes that will result in increased revenue or cost savings for an enterprise
Low	Slightly improves processes (for example, improved user experience) that will be difficult to translate into increased revenue or cost savings

Source: Gartner (August 2023)

Table 4: Maturity Levels

(Enlarged table in Appendix)



Maturity Levels	Status	Products/Vendors
Embryonic	In labs	None
Emerging	Commercialization by vendors Pilots and deployments by industry leaders	First generation High price Much customization
Adolescent	Maturing technology capabilities and process understanding Uptake beyond early adopters	Second generation Less customization
Early mainstream	Proven technology Vendors, technology and adoption rapidly evolving	Third generation More out-of-box methodologies
Mature mainstream	Robust technology Not much evolution in vendors or technology	Several dominant vendors
Legacy	Not appropriate for new developments Cost of migration constrains replacement	Maintenance revenue focus
Obsolete	Rarely used	Used/resale market only

Source: Gartner (August 2023)

Document Revision History

Hype Cycle for Software Engineering, 2022 - 1 August 2022

Hype Cycle for Software Engineering, 2021 - 27 July 2021

Recommended by the Authors

Some documents may not be available as part of your current Gartner subscription.

Understanding Gartner's Hype Cycles

Tool: Create Your Own Hype Cycle With Gartner's Hype Cycle Builder

2023 Strategic Roadmap for Becoming a World-Class Software Engineering Organization

Market Guide for AI-Augmented Software-Testing Tools

Emerging Tech: Generative AI Code Assistants Are Becoming Essential to Developer Experience

Use Value Streams to Design Service and Operating Models and Enable Application Composability

How Design, Development and Product Management Can Work Together Successfully

How to Make the Right Technology and Architecture Choices for Front-End Development

Demystifying Chaos Engineering for I&O

API Security: What You Need to Do to Protect Your APIs

Gartner

Table 1: Priority Matrix for Software Engineering, 2023

Benefit	*Years to Mainstream Adoption*			
↓	***Less Than 2 Years*** ↓	***2 - 5 Years*** ↓	***5 - 10 Years*** ↓	***More Than 10 Years*** ↓
Transformational	DevSecOps	AI-Augmented Software Engineering AI Coding Assistants Generative AI Observability Platform Engineering Site Reliability Engineering	Responsible AI Software Supply Chain Security	

Benefit	*Years to Mainstream Adoption*			
↓	***Less Than 2 Years*** ↓	***2 - 5 Years*** ↓	***5 - 10 Years*** ↓	***More Than 10 Years*** ↓
High	Citizen Developers Product-Centric Delivery Model	AI-Augmented Testing API Security Testing Continuous Quality Design Systems Feature Management GitOps Internal Developer Portal Open-Source Program Office Performance Engineering Prompt Engineering Secure Coding Training Software Bill of Materials Value Stream Management Platforms	AI Engineering Cloud Development Environments DesignOps Digital Immune System Green Software Engineering Observability-Driven Development Software Engineering Intelligence Platforms	
Moderate		DevOps Test Data Management	Behavior-Driven Development Event-Driven APIs Innersource	
Low				

Source: Gartner (August 2023)

Table 2: Hype Cycle Phases

Phase ↓	*Definition* ↓
Innovation Trigger	A breakthrough, public demonstration, product launch or other event generates significant media and industry interest.
Peak of Inflated Expectations	During this phase of overenthusiasm and unrealistic projections, a flurry of well-publicized activity by technology leaders results in some successes, but more failures, as the innovation is pushed to its limits. The only enterprises making money are conference organizers and content publishers.
Trough of Disillusionment	Because the innovation does not live up to its overinflated expectations, it rapidly becomes unfashionable. Media interest wanes, except for a few cautionary tales
Slope of Enlightenment	Focused experimentation and solid hard work by an increasingly diverse range of organizations lead to a true understanding of the innovation's applicability, risks and benefits. Commercial off-the-shelf methodologies and tools ease the development process.
Plateau of Productivity	The real-world benefits of the innovation are demonstrated and accepted. Tools and methodologies are increasingly stable as they enter their second and third generations. Growing numbers of organizations feel comfortable with the reduced level of risk; the rapid growth phase of adoption begins. Approximately 20% of the technology's target audience has adopted or is adopting the technology as it enters this phase.
Years to Mainstream Adoption	The time required for the innovation to reach the Plateau of Productivity.

Phase ↓	*Definition* ↓

Source: Gartner (August 2023)

Table 3: Benefit Ratings

Benefit Rating ↓	***Definition*** ↓
Transformational	Enables new ways of doing business across industries that will result in major shifts in industry dynamics
High	Enables new ways of performing horizontal or vertical processes that will result in significantly increased revenue or cost savings for an enterprise
Moderate	Provides incremental improvements to established processes that will result in increased revenue or cost savings for an enterprise
Low	Slightly improves processes (for example, improved user experience) that will be difficult to translate into increased revenue or cost savings

Source: Gartner (August 2023)

Table 4: Maturity Levels

Maturity Levels ↓	*Status* ↓	*Products/Vendors* ↓
Embryonic	In labs	None
Emerging	Commercialization by vendors Pilots and deployments by industry leaders	First generation High price Much customization
Adolescent	Maturing technology capabilities and process understanding Uptake beyond early adopters	Second generation Less customization
Early mainstream	Proven technology Vendors, technology and adoption rapidly evolving	Third generation More out-of-box methodologies
Mature mainstream	Robust technology Not much evolution in vendors or technology	Several dominant vendors
Legacy	Not appropriate for new developments Cost of migration constrains replacement	Maintenance revenue focus
Obsolete	Rarely used	Used/resale market only

Source: Gartner (August 2023)