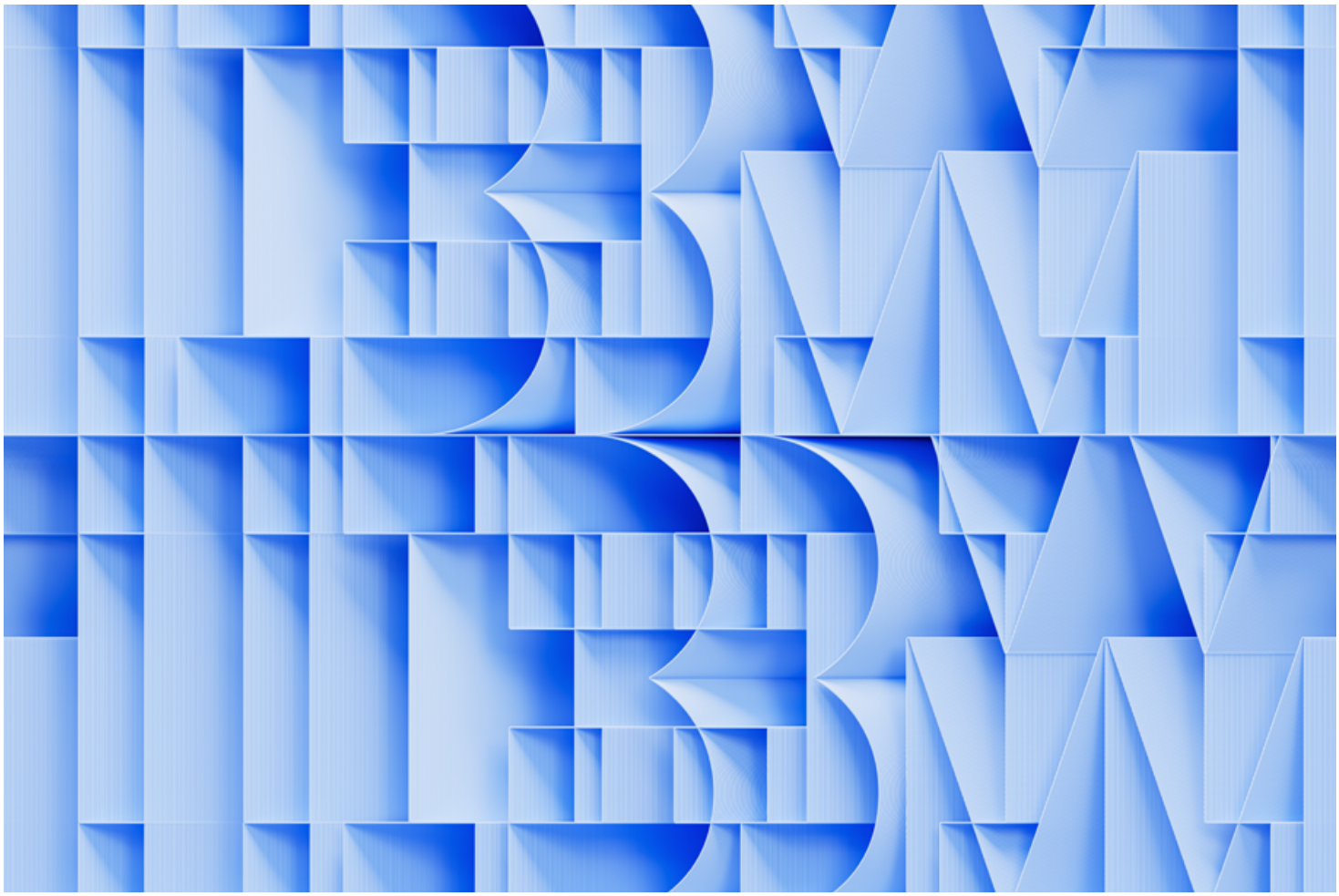


AI changed software development. Now it must change delivery.

As AI reshapes development, enterprises must adopt integrated systems that enable governed and scalable software delivery. Learn how organizations can make that shift.

IBM

Table of contents

01
The shift from tools to delivery systems

Today's enterprise software organizations are experiencing a significant shift in development output. Developers, and increasingly AI agents, are capable of sustained, high-output performance. Code generation accelerates, refactors complete in minutes (or even seconds) and test suites are automatically generated. The local constraints that once limited output are diminishing.

We have extended how long and how intensively teams and agents can operate. The critical question is whether this extended capacity is producing higher quality, system-level outcomes. For example:

- Are we increasing meaningful delivery, or just extending performance?
- Are we strengthening the enterprise system, or intensifying localized acceleration?
- Who exactly is the performance optimized for?

The enterprise now faces a paradox: individual productivity has never been higher, while organizational coherence is becoming increasingly critical.

Acceleration without alignment
The strain enterprises feel today is not resistance to innovation. It's structural misalignment. AI coding tools delivered a first-mover advantage at the local level, but tools are not systems. A tool optimizes momentum while a system governs direction and purpose.

As adoption scales, several considerations emerge:

- Productivity means different things at the individual, team and enterprise levels
- Developer identity varies across the organization
- SDLC models are unevenly harmonized
- Governance, security, compliance and integration demands sit downstream
- Agents are deployed without orchestration reflective of enterprise nuance
- Automation scales faster than integration mechanisms mature

This creates a gap between local acceleration and enterprise outcomes.

Developer identity precedes developer productivity
Before an enterprise can optimize software delivery, it must understand the identity of its developers. Not all teams are solving the same problem, and they should not be measured or equipped as if they are.

Across large organizations, differences are structural:

- Platform teams optimize differently than product teams
- Regulated environments optimize differently than internal tooling groups
- Automation tolerance varies
- Governance expectations differ
- Velocity signals vary
- Risk posture varies

These differences shape how work should be executed and evaluated. Without alignment between developer identity, metrics and SDLC structure, increased agent capability may accelerate divergence rather than convergence.

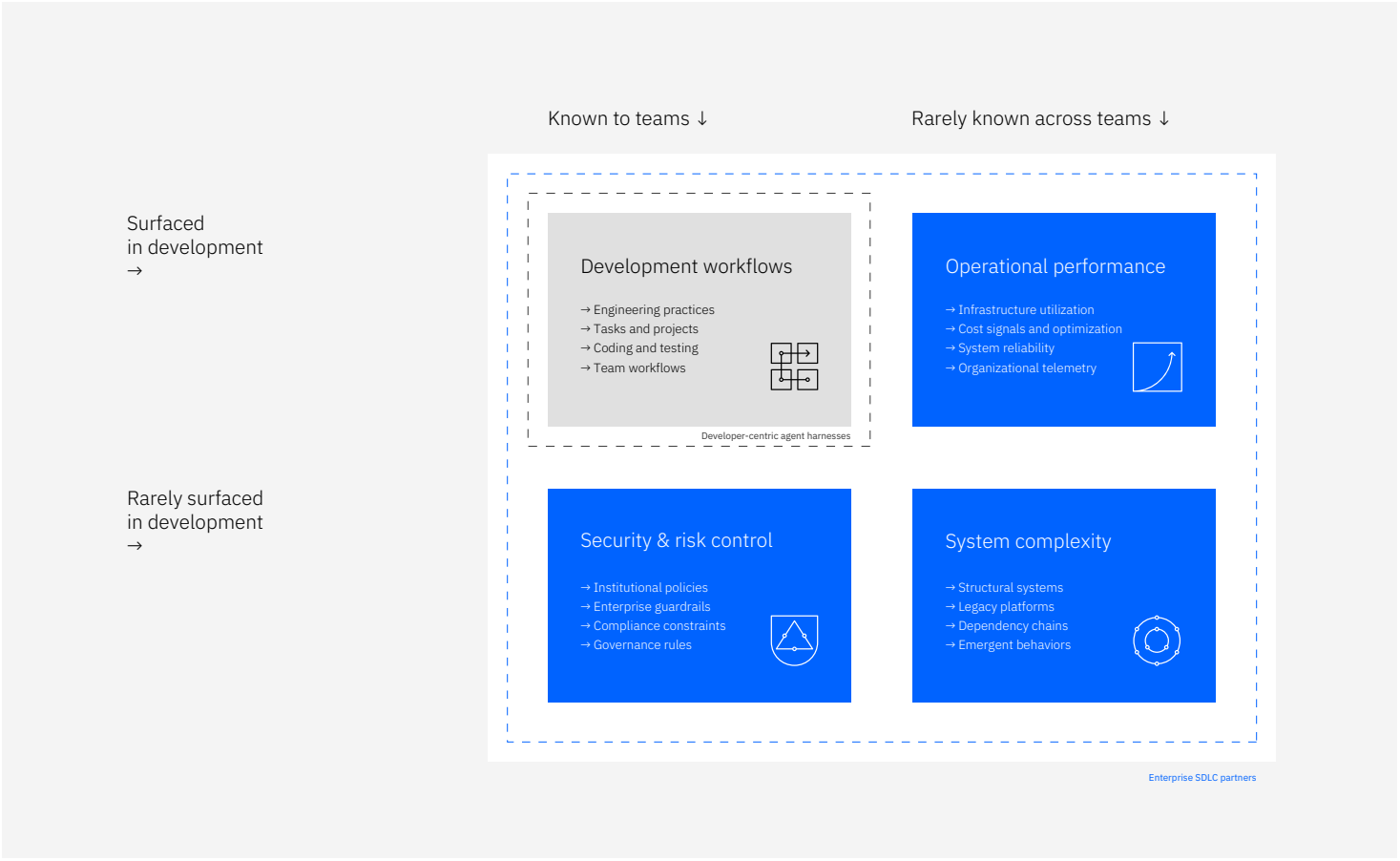
The durable advantage: From tools to systems
The enterprise conversation is evolving. It's no longer asking, "How do we give developers performant AI tools?" Scaling sustainably requires deliberate coordination across teams, workflows and controls. Scaling AI across the SDLC demands more than model capability. It requires delivery orchestration that has:

- Development systems aligned to enterprise policy
- Governance-aware automation embedded upstream
- System-level observability across human and agent actions
- Cognitive load balancing and decision-support layers
- Cross-functional integration by design
- Resilience engineered into deployment pathways
- Cost and tradeoff transparency

The organizations that operationalize end-to-end delivery systems will spend less time managing tools and more time delivering outcomes.

The real challenge facing enterprise AI isn't capability—it's coordination.

How do we integrate AI into the full delivery system, from idea to production, governance to audit or code to consequence?



Enterprise software delivery spans multiple layers of knowledge and visibility. Developer-centric agent harnesses (gray square) primarily operate within development workflows where tasks are well-defined and code context is explicit. Enterprise delivery systems or SDLC partners (blue and gray squares) must operate across a broader landscape that includes operational signals, governance constraints and architectural complexity.

The advent of the SDLC partner

It's encouraging to see compelling providers recognize the importance of moving up the enterprise stack solutions towards more enterprise-aware systems. The evolution from tool to platform is a healthy and necessary progression.

Where we believe the conversation must continue to mature is in how these systems integrate more holistically into an enterprise delivery system.

We believe the future of delivery is defined by the emergence of what we call an SDLC partner, or a system that:

- Understands developer identity
- Harmonizes productivity metrics across levels
- Integrates governance and downstream impact by design
- Orchestrates agents with control rather than unleashing them
- Connects into the wider enterprise substrate
- Evolves alongside the organization but ensures consistency

A tool can accelerate work, but a partner aligns this work with larger enterprise goals. The opportunity ahead is not simply to move faster, it's to move coherently.

02
How should we deliver?

We can now generate, refactor, test and deploy at a sustained velocity. For many teams, technical capacity is no longer the constraint. Collaboration between humans and machines has expanded what's possible. The strategic question is no longer whether we can keep moving, but how that movement is structured, governed and aligned to enterprise purpose.

Several themes surface repeatedly as organizations move from isolated tools to integrated delivery systems. Four warrant focused attention.

Enabling teams: Individual productivity rises, but enterprise value depends on coordinated team execution

Shift-left security: Security must evolve from oversight to embedded, real-time delivery control

Modernization at scale: Legacy complexity persists, requiring systems that adapt across environments

Cost control: AI introduces variable cost models requiring active management and system-level optimization

01

Empowering teams

AI-assisted development is shifting from simply accelerating individual coding speed to improving end-to-end workflow coherence across application teams, platform engineering, SRE, security and modernization. Organizations should adopt systems that orchestrate human talent and agents across the SDLC through a context-aware collaboration layer.

Identity-aware workflows
Workflows should adapt to developer identity and role, so capabilities, guardrails and automation align with responsibilities. Embedded standards and documentation can also raise the productivity of junior engineers, new hires and cross-functional contributors. The result is role-aligned execution, reduced onboarding time, improved developer satisfaction, actionable knowledge transfer and consistent productivity gains without disrupting established delivery practices.

Human-governed agent workflows
As agents take on more complex tasks, developers should retain strategic oversight through structured workflows in which agents propose changes, developers review intent and impact while orchestration enforces governance. Planning previews, diffs and structured approvals help balance automation with oversight without adding unnecessary burden.

Cross-system context awareness
At the same time, systems should reason across repositories, services, APIs and infrastructure, integrating code context with operational telemetry so developers can understand cross-system dependencies and impacts. By integrating insights across codebases, environments and operational telemetry, AI systems can help teams understand how changes affect broader system behavior.

Developer knowledge and flow
Because enterprise environments often rely on decades of institutional knowledge embedded in legacy applications, documentation and developer expertise, AI systems should extract, document and operationalize this knowledge to make it accessible across teams. By bringing this context directly into the development environment, AI reduces cognitive load and avoids the “Alt-Tab tax,” preserving flow and accelerating decision-making.

02

Shift-left security

Historically, security validation occurred late in the delivery lifecycle, delaying releases as teams reconciled policy violations, infrastructure risks and compliance requirements.

In AI-assisted development, security must become architectural and be embedded proactively. Modern DevSecOps practices shift security earlier through static analysis, dynamic validation and policy-as-code. AI development systems extend this shift by embedding security awareness directly into development workflows.

↓

Pre-merge security and policy-as-code governance
AI development platforms should evaluate generated changes during authoring and pull requests, enforcing architectural guardrails, security rules and compliance requirements before code reaches shared repositories. Embedding these checks earlier in the development lifecycle reduces the risk of insecure patterns propagating through the delivery pipeline while maintaining development momentum.

At the same time, policy-as-code frameworks should provide version-controlled, machine-readable governance for code, infrastructure and architectural decisions across teams and pipelines. By managing security policies as version-controlled artifacts that propagate across teams and pipelines, organizations can enforce governance consistently without introducing operational friction.

Continuous code and infrastructure security validation
Continuous validation—combining static and dynamic analysis within IDEs and CI/CD—should surface vulnerabilities and misconfigurations as they emerge, turning security into an inherent property of software delivery rather than a periodic audit. Safeguards should detect secrets, API keys and confidential data in both generated artifacts and the contextual information AI systems consume.

This approach ensures that vulnerabilities and misconfigurations are detected during development rather than discovered after deployment, allowing organizations to maintain delivery velocity while strengthening security posture.

Setting boundaries and end-to-end traceability
AI-aware threat protections can mitigate prompt injection attacks, malicious context manipulation and unsafe generation patterns. Role-based access control helps ensure agents act within enterprise identity and permission boundaries. Comprehensive audit trails should capture prompts, artifacts, approvals and deployments to maintain clear traceability of both human and automated actions to support incident investigation, regulatory reporting and accountability. Together, these capabilities help identify and address violations pre-merge, enforce consistent governance, reduce exposure risk and maintain transparent, compliant delivery at scale.

03

Modernization at scale

AI has the potential to dramatically accelerate modernization, but only when it works within a structured delivery system that preserves existing behavior and validates outcomes.

Dependency mapping across legacy estates
Effective modernization requires visibility into how systems behave today, how components depend on each other and how change can occur without disrupting production. Enterprise AI development systems should map dependencies across repositories, services, APIs and data flows, enabling cross-file and cross-repository reasoning that reveals interaction patterns and potential risk before work begins. They should generate tests, behavioral comparisons and validation artifacts to preserve institutional logic, allowing teams to review proposed changes and confirm functional equivalence throughout refactoring or migration.

Risk-scored modernization sequencing
Safe modernization rarely happens in a single step. Enterprise AI systems should support phased refactoring strategies where structural improvements occur incrementally and are validated at each stage.

Risk-scored modernization sequencing should target low-risk components early, building confidence before tackling more complex areas. Along the way, AI tools should capture architectural knowledge by summarizing legacy modules, explaining inherited logic, documenting transformation intent and recording decisions.

Cross-system impact awareness
Cross-system impacts should be assessed in advance by analyzing service contracts and downstream dependencies, while compliance workflows and audit trails maintain a verifiable record of what changed, why and how it was validated. Sustained modernization requires using AI within a disciplined framework that reveals system dependencies, preserves institutional logic, sequences change safely and ensures every transformation is validated and understood.

04

Cost control

As enterprises embed AI across the software development lifecycle, governing the economics of delivery becomes just as important as accelerating it. Beyond the raw cost of models, organizations must manage the broader cost drivers of software creation—from code generation and testing to infrastructure shifts and ongoing operational workflows.

An enterprise AI development system should function as an economic control layer by making resource consumption visible, aligning workloads with the appropriate models and infrastructure, and surfacing cost performance tradeoffs before execution.

When cost awareness is built into development workflows, AI shifts from unpredictable spend to improved economic efficiency.

Intelligent model routing
Cost control requires intelligent model routing, budget enforcement, workflow-level efficiency tracking and deep observability into resource consumption across teams and pipelines. Routine tasks can be steered automatically to lightweight models while more complex reasoning, like architectural reasoning or multi-file analysis, may leverage higher-capability models. This allows organizations to continuously optimize performance and cost without disrupting developer workflows.

Budget threshold enforcement
Budget thresholds and policy controls help provide predictable economic boundaries, while lifecycle-focused efficiency metrics help organizations evaluate AI's true impact beyond token usage. With granular cost attribution and real-time economic feedback loops, teams can proactively understand the cost implications of large-scale refactors, multi-agent tasks or infrastructure analyses before they run. This in turn allows organizations to scale adoption safely without risking uncontrolled consumption.

Efficiency and cost optimizations
As AI tools mature, enterprises should evaluate solutions that extend beyond code generation to support the broader operational workflows developers rely on. Organizations should prioritize systems that can analyze system telemetry and development signals to help teams reduce operational waste across development and infrastructure environments.

Capabilities that connect development workflows with operational insights will become increasingly important as teams manage growing system complexity while maintaining delivery reliability and efficiency. These insights should also provide the ability to track improvements across the delivery lifecycle, including rework and defect reduction, faster test generation and improvements in time-to-merge or deployment stability.

What 2025 adoption revealed	What enterprise systems must deliver next
AI tools dramatically increased individual developer throughput	Organizational delivery productivity becomes the new optimization target
Human + AI pairing improved local development workflows	Team + AI pairing becomes a managed delivery system
Code generation accelerated feature implementation	End-to-end delivery orchestration governs how work moves to production and maintenance
Security review processes needed to keep pace with AI output	AI-driven security can be proactive, with compliance embedded throughout the SDLC
Governance models lagged behind automation capabilities	Governance defines automation as an embraced feature from dev teams, not a hindrance
Code modernization efforts gained new momentum through AI assistance	Continuous modernization integrated into delivery systems for more end-to-end assistance
Institutional knowledge surfaced through AI-assisted code understanding	Architectural context becomes a shared enterprise capability
AI consumption introduced new economic considerations for development teams	Economic delivery governance manages the cost of acceleration
Development tools began interacting with infrastructure and operational systems	Development platforms incorporate operational and infrastructure context

03
The enterprise delivery decision: Build, adopt or orchestrate?

As enterprises move from AI-assisted development to operational delivery, a practical question begins to surface: Should organizations build their own AI-enabled delivery systems, adopt one designed for enterprise environments, or combine elements of both?

Practically speaking, most enterprises will do a combination. The market has produced powerful AI tools and agent harnesses that dramatically improve local developer productivity. But as adoption scales, organizations increasingly require systems that govern how those agents operate across the broader software delivery lifecycle.

Organizations evaluating AI-enabled development systems typically encounter three architectural paths:

- Build internally: Assembling models, agent harnesses, governance layers and delivery workflows into a custom system
- Assemble an AI toolchain: Adopting multiple AI tools while building internal orchestration, governance and integration across them
- Integrate an SDLC partner: Deploying a system designed to orchestrate AI across enterprise delivery

Most enterprises will not simply buy a tool and plug it in. At the same time, few organizations can realistically build and maintain every layer of a modern delivery substrate themselves.

What initially appears to be an AI tool decision quickly becomes an enterprise systems architecture project. Operating in isolation is not enough. Systems must function as a coherent whole. Without that integration, organizations risk accelerating local development activity without improving enterprise delivery outcomes.

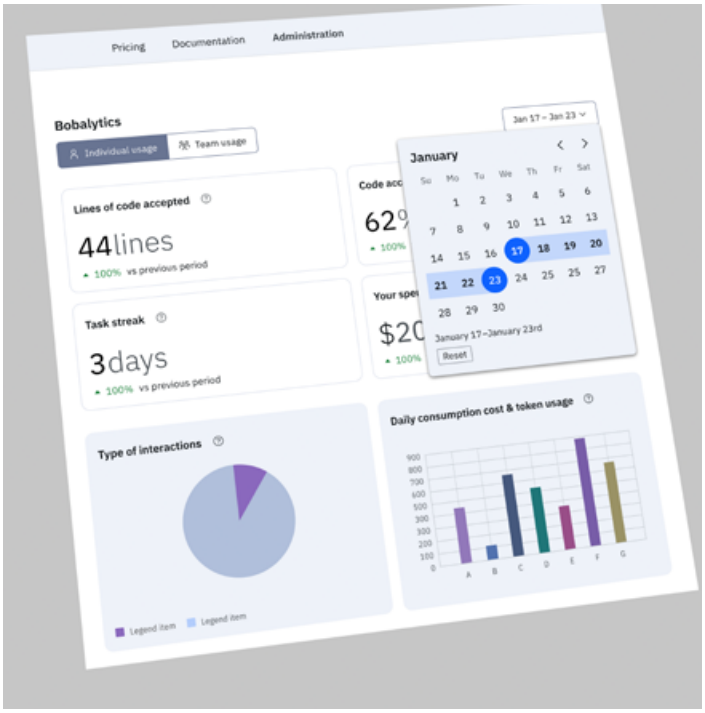
Enterprises that succeed in the next era of software development will not simply move faster—they will move coherently.

They will operate systems that:

- Orchestrate human and agent workflows
- Surface operational and architectural signals
- Embed governance directly into development
- Align developer productivity with enterprise outcomes

The question facing enterprise leaders is no longer whether AI will accelerate software development and productivity. That transformation already happened. For many enterprises, the answer will be the system that helps them move not just faster, but together. Increasingly, that system will look like an SDLC partner.

04
Adopting your SDLC partner



IBM designed an SDLC partner, IBM Bob, in response to what enterprises consistently told us they needed. Across industries, organizations are not simply asking for faster code generation. They are asking for systems that understand the realities of enterprise delivery.

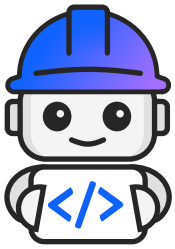
They need:

- A system that orchestrates human developers and AI agents together
- Governance and security embedded directly in development workflows
- AI that operates across complex architectures and legacy systems
- Visibility into cost, operational signals and delivery outcomes

Existing tools solved important parts of the problem, but enterprises were still responsible for assembling the full system. Rather than requiring enterprises to assemble their own system from a variety of models, harnesses, governance layers and orchestration tooling, Bob is designed as an end-to-end SDLC partner that integrates these capabilities directly into the development system.

The organizations that outperform in the next era of software delivery will be the ones that shift from AI-accelerated activity to AI-orchestrated delivery systems. Bob was built to help enterprises make that shift—moving from isolated tooling to a coherent, governed SDLC partner.

Learn more about IBM Bob →



Real-world experiences

Customers who worked with IBM to test the beta version of IBM Bob.

We achieved 100% refactored code. Everything I thought would be a problem, Bob solved. The old .NET SOAP service was completely refactored to modern REST API with clean architecture.

Veran Pokornić
Solution Architect
APIS IT

Working with IBM through Bob was a highly collaborative experience. The framework brought structure, speed, and confidence to the engagement, enabling us to deliver measurable value for our business and our clients.

Saireshan Govender
Group CEO
Blue Pearl

Bob helped solve several challenges we were facing—especially around lack of documentation, large code bases and older code. We were able to take that and give it to brand new developers, who could quickly understand and start contributing.

Todd Stewart
Application Architect
Heartland Co-op

Bob represents a meaningful step forward in how we think about AI-assisted development.
It enables faster progress while still maintaining a clear sense of human accountability.

Uwe Graf
Modernization Architect
EasiRun Europa GmbH