Reducing the Software Delivery Lifecycle to Increase Value



It's Time to Speed Up Time to Market

Research and practice synthesis | Prepared for university–industry discussion

Keywords: software delivery lifecycle; end-to-end lead time; value stream management; continuous delivery; AI-assisted development; DORA; software quality; governance.

Abstract

This paper examines how software organizations can reduce the total elapsed time between an approved business need and a verified production outcome. The analysis emphasizes end-to-end flow rather than local coding productivity, integrating value-stream management, small batches, automation, continuous delivery, and risk-based human oversight. Research from DORA, Microsoft Research, peer-reviewed software engineering literature, METR, and security standards provides a basis for evaluating both opportunities and constraints. The evidence suggests that AI assistance can improve task-level productivity in some contexts while adding friction in others. Accordingly, the research proposes a measurable lifecycle-optimization framework assessed through lead time, delivered value, reliability, engineering effort, and security outcomes.

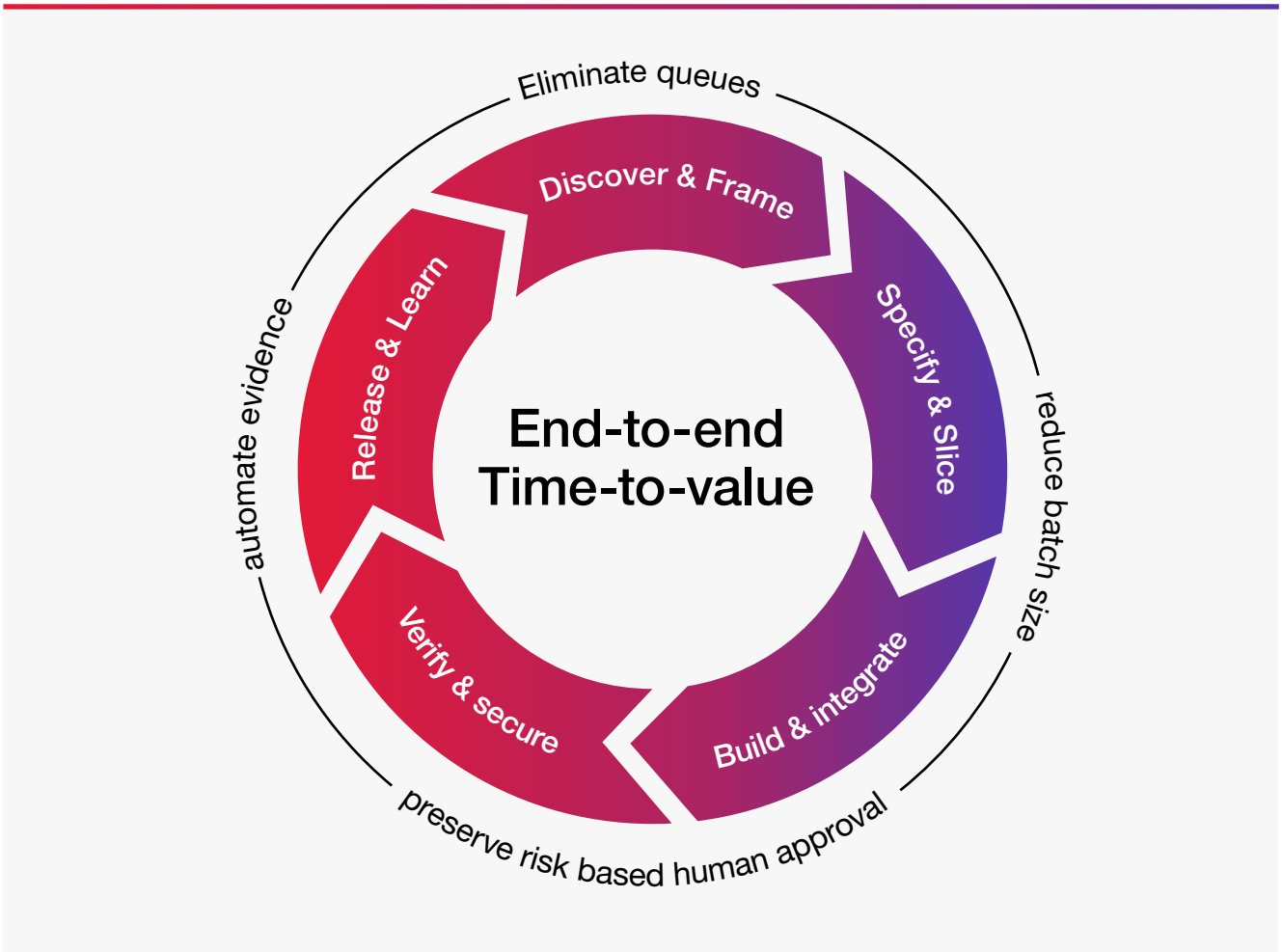


Figure 1. Proposed end-to-end delivery lifecycle. Author-created conceptual diagram.

1. Introduction and research question

Software delivery performance is not reducible to the speed at which code is generated. Business demand can spend substantial elapsed time in discovery queues, clarification, dependencies, review, test environments, compliance approval and release scheduling. Optimizing a single step may simply move the bottleneck downstream. The principal research question is: which organizational, engineering and AI-enabled practices can reduce request-to-value lead time without degrading safety, reliability, maintainability or developer well-being?

The central proposition is that reducing lifecycle duration requires coordinated improvements across demand intake, decision-making, implementation, integration, verification, release, and post-release feedback. Research on flow and delivery performance motivates smaller batch sizes, reduced work in progress, earlier validation, reliable automation and explicit risk controls. These recommendations are synthesis and proposed interventions, not claims that any single cadence or tool guarantees faster
delivery [1–4].

2. Conceptual foundations

2.1 Flow efficiency versus local productivity

A delivery lifecycle combines active work and waiting. The appropriate target is total elapsed time from accepted request to verified value, with a complementary accounting of active effort and post-release learning. Small batches, controlled work-in-progress, explicit entry and exit policies, and faster feedback loops can help reduce queues. In this paper, “flow efficiency” is defined operationally as active processing time divided by total elapsed lifecycle time; it should be interpreted cautiously when activities overlap.

2.2 What DORA research supports

DORA separates delivery throughput from stability, rather than treating faster deployment as success in isolation. Its 2024 research cautioned that AI adoption could improve individual experience while coinciding with poorer software delivery stability and throughput; the 2025 research frames AI as an amplifier of an organization's pre-existing capabilities and reports the importance of platform quality, testing and value-stream management [2–4]. These findings support system-level measurement, not a universal claim that AI agents shorten end-to-end delivery.

2.3 Mixed causal evidence on AI coding speed

Field experiments reported by Cui and colleagues, covering 4,867 developers across three companies, found a pooled 26.08% increase in completed tasks with AI coding assistance [5]. In contrast, METR's randomized trial with 16 experienced open-source developers and 246 tasks found a 19% increase in completion time with early-2025 AI tools in that environment [6]. Outcomes differ by task context, baseline expertise, tool maturity and evaluation method. Neither study directly verifies a fully autonomous 24-hour enterprise lifecycle.

3. Delivery delay taxonomy and levers

Lifecycle source of delay	Targeted intervention	Boundary / control
Ambiguous intent and late decisions	Outcome hypotheses; thin vertical slices; early design constraints	Product owner owns acceptance criteria
Queueing across specialist teams	WIP limits; shared platforms; cross-functional pods	Protected review capacity
Build/integration bottlenecks	CI; trunk-based development; ephemeral environments	Reproducible pipelines and rollback
Manual or serial quality checks	Automated unit/ integration/security evidence	Risk-based sign-off for sensitive changes
Release windows and approvals	Progressive delivery; policy-as-code; feature flags	Change-risk classification
Rework after production	Observability; fast feedback; incident reviews	Reliability and security floors

4. A redesigned delivery model: continuous micro-delivery

The proposed model is continuous, evidence-based delivery: business needs are decomposed into independently testable increments; delivery tooling and AI assistance reduce repetitive work; automated controls provide rapid feedback; and human decision-makers retain accountability for risk, architecture and production acceptance. The operating rhythm should adapt to task complexity and service criticality rather than impose a universal release-time target.

Stage	Suggested service-level design	Human accountability
Frame	Record outcome, owner, risk tier, success measure and dependencies	Product owner / business sponsor
Prepare	Approve minimal vertical slice and executable acceptance tests	Product + architecture
Execute	Agent-assisted implementation, CI, testing and traceable change summary	Technical lead for exceptions
Assure	Automated quality/ security checks plus conditional manual gate	Security / compliance / quality
Release	Progressive deployment; rollback readiness; production telemetry	Authorized release owner
Learn	Measure adoption/ value, incidents, rework and next-cycle learning	Business + engineering

For low-risk changes in modular systems, automated verification and progressive release can shorten time between approved intent and deployment. Ambiguous requirements, sensitive dependencies, schema changes, privacy or security risks, and irreversible actions require explicitly defined escalation paths and proportionate human approval. Such controls align with broader software assurance principles and trustworthy-AI risk management guidance [8–10].

5. Illustrative delivery-time scenario

Figure 2 provides a transparent hypothetical scenario to illustrate bottleneck removal; its figures are assumptions, not research findings or CGI operating data. A baseline lifecycle of 27 calendar days falls to 13 only if reductions in waiting, testing and release latency actually occur. Eliminating six days of coding effort alone would not deliver the same effect.

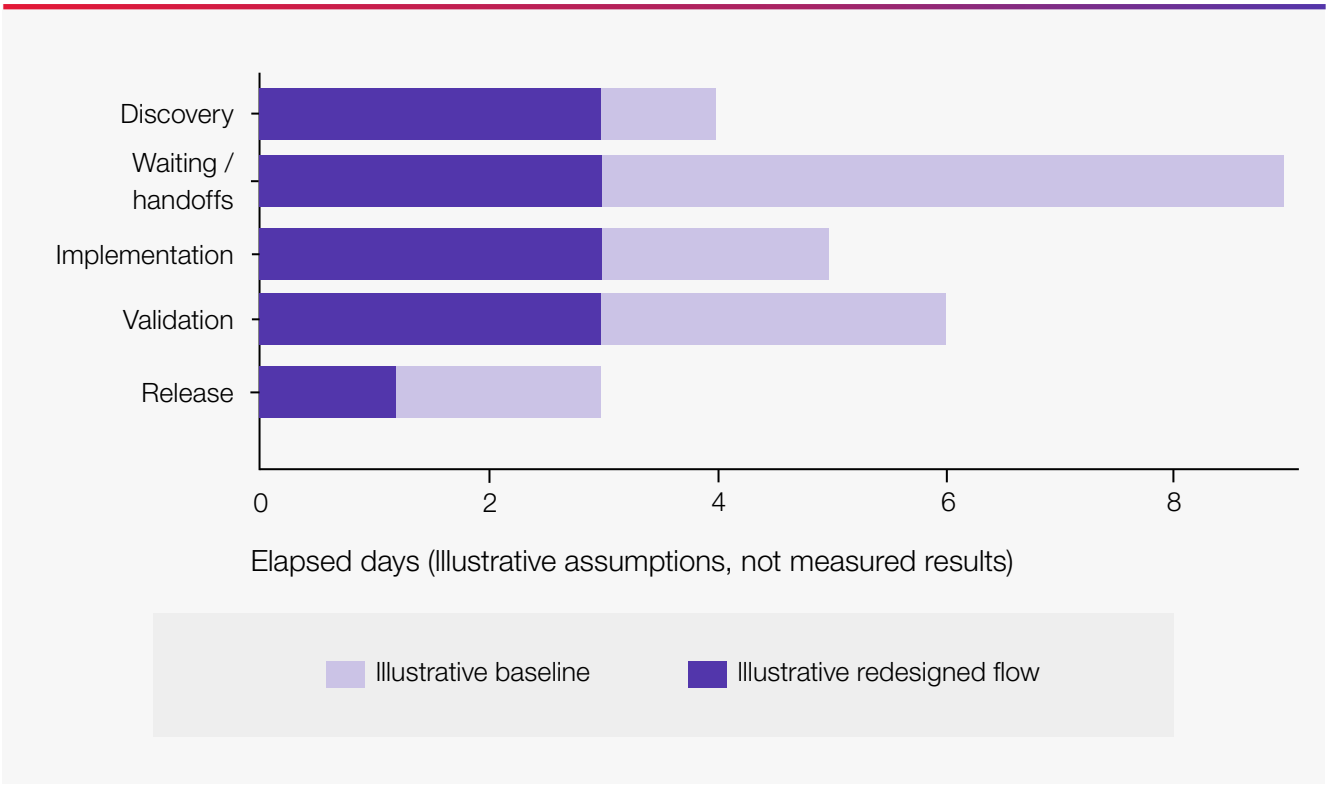


Figure 2. Hypothetical scenario, not empirically observed performance. Overlaid bars compare stage durations; the totals are 27 and 13 days.



6. Measurement and empirical design

The primary dependent variable is median and 85th-percentile request-to-production lead time, with a separate outcome-to-value interval when business effects lag release. Secondary measures include change lead time, deployment frequency, rework rate, change failure rate, failed-deployment recovery time, engineering effort per accepted change, WIP, review queue time, customer outcome attainment and team well-being. Productivity should be multidimensional, consistent with the SPACE framework [7].

Measure	Operational definition	Reason
End-to-end lead time	Accepted need → production verification, p50/p85	Primary lifecycle outcome
Waiting share	Time blocked/in queue ÷ elapsed time	Identifies structural bottlenecks
Value realization	Outcome metric change after release	Prevents output-only optimization
Delivery stability	Change failure, rework and recovery	Protects service quality
Human load	Review, supervision, incident and rework hours	Captures hidden agent costs
Cost per accepted change	Total delivery cost ÷ accepted changes	Tests economic viability

Recommended study: select two comparable low-risk services or matched workstreams; gather 4–6 weeks of baseline data; deploy the new process to one group for 6–8 weeks while maintaining a comparison group; stratify by task size, dependency, criticality and AI-tool experience. Use interrupted time-series or difference-in-differences analysis where assumptions are met, with qualitative interviews to explain bottlenecks. Pre-register hypotheses, exclusions and guardrail thresholds. The treatment should fail the success test if speed improves while serious incidents, defect escape, security exposure or unsustainable human workload increase.



7. Hypotheses and limitations

H1: Thin slicing and work-in-progress controls reduce median request-to-production lead time. H2: AI-supported implementation reduces active engineering time for suitable tasks, with heterogeneous effects. H3: Automated testing and release evidence reduce queue time without increasing change failures. H4: Risk-tiered approvals outperform undifferentiated manual gates on elapsed delivery time while preserving quality. H5: Value-stream improvements have a larger association with end-to-end lead-time reduction than local coding-speed improvements alone.

Limitations include confounding organizational changes, heterogeneous services, uneven task difficulty, value outcomes that become visible only after release, novelty effects, and incomplete measurement of hidden review or rework effort. AI-generated code also introduces verification, security, and maintainability burdens that may offset apparent execution gains.

8. Conclusion

Reducing the software delivery lifecycle requires shortening the entire path from intent to verified value, especially waiting, handoffs and rework, rather than merely accelerating code creation. A combination of value-stream management, small batches, continuous integration, automated quality evidence, progressive delivery and governed AI assistance provides a credible and testable improvement programme. Success should be established empirically through simultaneous gains in lead time and delivered value without deterioration in reliability, security or total human workload.

References and further institutional research

[1] DORA / Google Cloud. (2025). State of AI-assisted Software Development. Open source

[2] DORA / Google Cloud. (2024). Accelerate State of DevOps Report; DORA publications archive. Open source

[3] Forsgren, N., et al. (2021). The SPACE of Developer Productivity. ACM Queue. Open source

[4] Google Cloud / DORA. Software delivery and operational performance research resources. Open source

[5] Cui, Z., et al. (2026). The Effects of Generative AI on High-Skilled Work: Evidence from Three Field Experiments with Software Developers. Management Science. Open source

[6] METR. (2025). Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. Open source

[7] Carnegie Mellon University, Software Engineering Institute. DevSecOps and software engineering research. Open source

[8] NIST. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). Open source

[9] OWASP. GenAI Security Project and guidance. Open source

[10] NIST. AI Risk Management Framework resources and implementation guidance. Open source

[11] Massachusetts Institute of Technology. Research institution and author affiliation in the cited field-experiment study. Open source

[12] University of Pennsylvania, Wharton School. Academic affiliation in the cited field-experiment study. Open source

[13] Microsoft Research. Software engineering and AI-assisted development research. Open source

Organizations, institutions and universities for further study

The following sources are relevant to extending and independently evaluating this research: Google DORA (software delivery performance and AI adoption); Microsoft Research (field evidence on developer productivity); the Software Engineering Institute at Carnegie Mellon University (software engineering and DevSecOps); MIT and the University of Pennsylvania (academic affiliations represented among co-authors of the cited controlled field-experiment paper); ACM (developer-productivity research); METR (independent empirical testing); NIST (AI risk management); and OWASP (security guidance). Inclusion as a reference does not suggest endorsement, participation or a formal research partnership.



Author



Miguel Coelho
Vice President
Consulting Delivery

About CGI

Insights you can act on

Founded in 1976, CGI is among the largest IT and business consulting services firms in the world.

We are insights-driven and outcomes-focused to help accelerate returns on your investments. Across hundreds of locations worldwide, we provide comprehensive, scalable and sustainable IT and business consulting services that are informed globally and delivered locally.

For more information
Visit cgi.com
Email us at info@cgi.com

© 2026 CGI Inc.