



The Context Continuum:

Engineering trust across the AI-powered SDLC



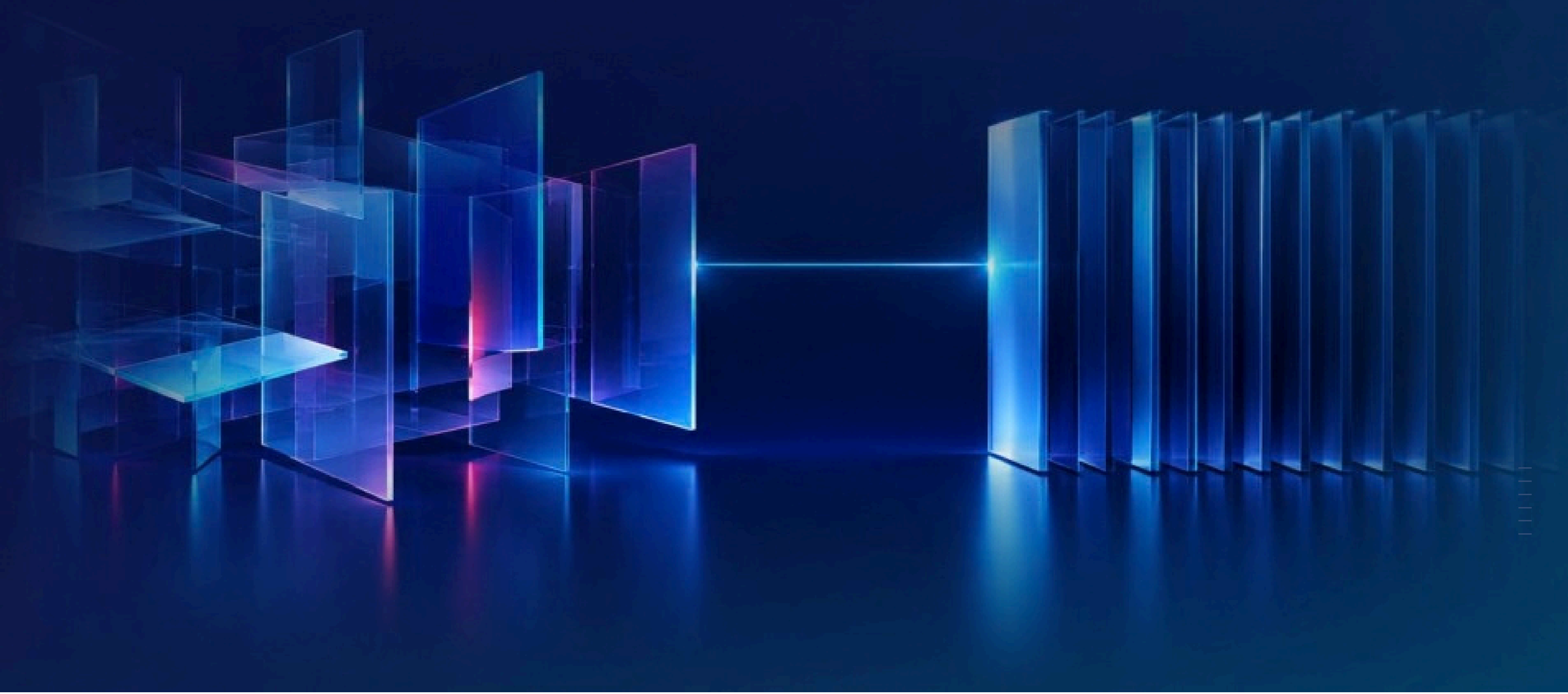
The context gap: why AI speed is outrunning banking control.

AI is changing software engineering at remarkable speed. Requirements can be synthesized from thousands of pages of documentation. Designs can be explored in hours. Code, test cases and deployment scripts can be generated in minutes.

McKinsey's November 2025 survey of nearly 300 publicly traded companies found the highest-performing AI-driven software organizations achieving 16–30% gains in team productivity, customer experience and time to market, and 31–45% gains in software quality, while most adopters see far less. DX's most recent telemetry across 135,000+ developers puts AI tool adoption at 93%, with developers self-reporting an average of 3.9 hours saved per week.

But faster activity is not the same as faster, safer delivery. Google DORA's 2024 research found that a 25% increase in AI adoption was associated with better documentation, code quality and review speed, but also came with a ~7.2% decline in delivery stability. Its 2025 research sharpened the point: AI amplifies what is already present. Strong platforms, clear workflows and fast feedback loops become force multipliers, while fragmented knowledge, tightly coupled architectures and weak controls become sources of instability.

This is the emerging context gap. AI agents can execute an instruction, but they do not automatically inherit the business reasoning, domain rules, architectural decisions, regulatory obligations, security controls and operational learning that give the instruction meaning. When each agent works from a different version of reality, the organization may produce more artefacts and more code, but also create more inconsistency, rework and risk.





Context engineering closes that gap. It creates a governed continuum of intent, knowledge, constraints, evidence and memory from demand to production. It is not a larger prompt or another repository. It is an operating system for AI-powered delivery - one that assembles the right context for each task, preserves provenance, enforces policy, evaluates outcomes and learns from production.

For banks, this is more than an engineering improvement. Software encodes credit policy, pricing, fraud controls, customer consent and regulatory obligations. Context engineering is therefore the discipline that allows AI-driven speed to coexist with control, explainability and trust. We have seen this play out directly with a global banking leader: reconstructing intent from undocumented legacy banking screens, rather than converting code line by line, cut per-screen effort by more than 80% (from about a week to under a day) and reduced program effort from about 1,000 specialist engineer-weeks to 500–700, without loosening audit control.

This paper is written for the people who must both mandate that shift and operate it. The CIO decides that the next investment is context, not another model. CRO, CISO and audit need provenance, dual control and a third-party boundary they can examine. Directors and function heads - product, architecture, engineering, quality, operations, risk and change, need named ownership, a contract at each hand-off, and a first journey they can run.

Introduction: Copilots raised output. They did not raise assurance.

The first wave of AI-assisted development focused on the developer. Copilots reduced the effort required to draft functions, explain unfamiliar code and create unit tests. The next wave is broader: specialized agents are beginning to participate across requirements, design, development, quality engineering, release and operations. The SDLC is moving from human-led work augmented by isolated tools to human-governed delivery orchestrated by multiple agents.

Adoption is no longer the constraint. Stack Overflow's 2025 survey found that 84% of developers used or planned to use AI tools, yet only 29% trusted AI outputs to be accurate. The same research found that 66% were frustrated by solutions that were "almost right", while 45% said debugging AI-generated code was more time-consuming. This is the underlying issue of the AI-powered SDLC: output is accelerating faster than confidence.

The answer is not to slow AI down, rather it is to give AI a reliable understanding of the enterprise in which it must operate. Models provide intelligence; tools provide the ability to act; context provides the basis for acting correctly. Without that basis, autonomy merely increases the speed and scale at which incomplete assumptions travel through delivery. If teams are already deploying copilots and agents, the next conversation is not which model to add. It is how much of the institution's context those agents can actually see.



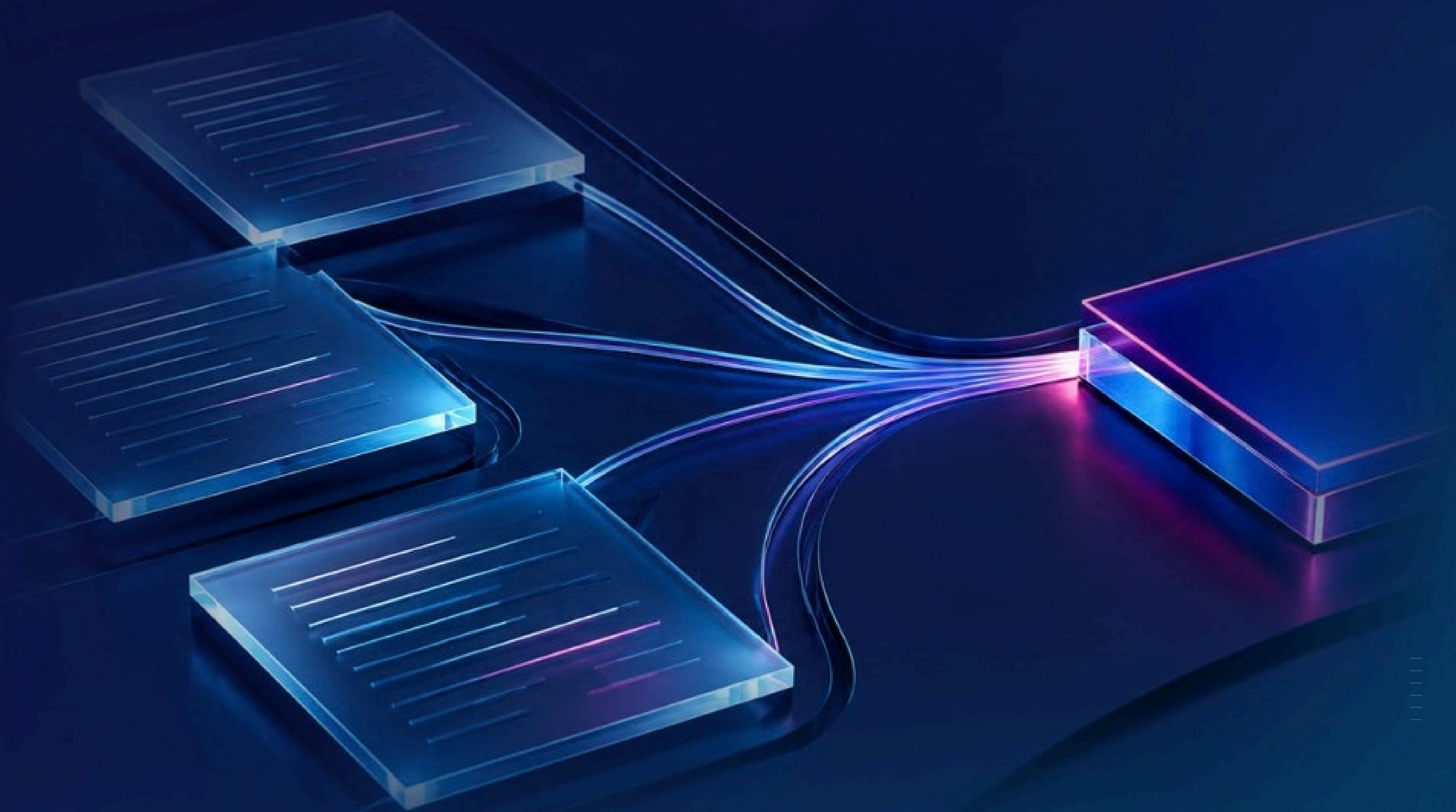
1. Faster stages, slower institutions

AI now touches every stage of software delivery. The value does not come from replacing one role or automating one task. It comes from shortening the distance between an idea, an engineering decision and a production outcome. At each stage the failure mode is the same: the agent can produce an artefact, but not inherit the bank.

1.1 Requirements: synthesis is not intent

Requirements work has traditionally depended on business analysts interviewing stakeholders, reading process documents, reconciling competing views and translating loosely expressed needs into structured specifications. AI can compress much of this discovery effort. Agents can parse interview transcripts, product manuals, service tickets, policy documents and process data; identify recurring needs; surface contradictions; and convert the material into functional requirements, epics, user stories and acceptance criteria.

The real advantage is not document generation, but synthesis at scale. A requirements agent can connect a customer complaint in a call-centre transcript with an exception in an operations manual and a rule in a compliance policy. It can then flag what is missing before the requirement reaches design. Human product owners remain accountable for intent and priority, but they work from a stronger evidence base.



1.2 Design: plausible is not admissible



At the design stage, AI can translate functional requirements into alternative architectural patterns, interface models, data schemas, entity relationships and API specifications. It can compare options against approved reference architectures, inspect logical flows for bottlenecks and security exposure, and convert wireframes into working prototypes for early validation.

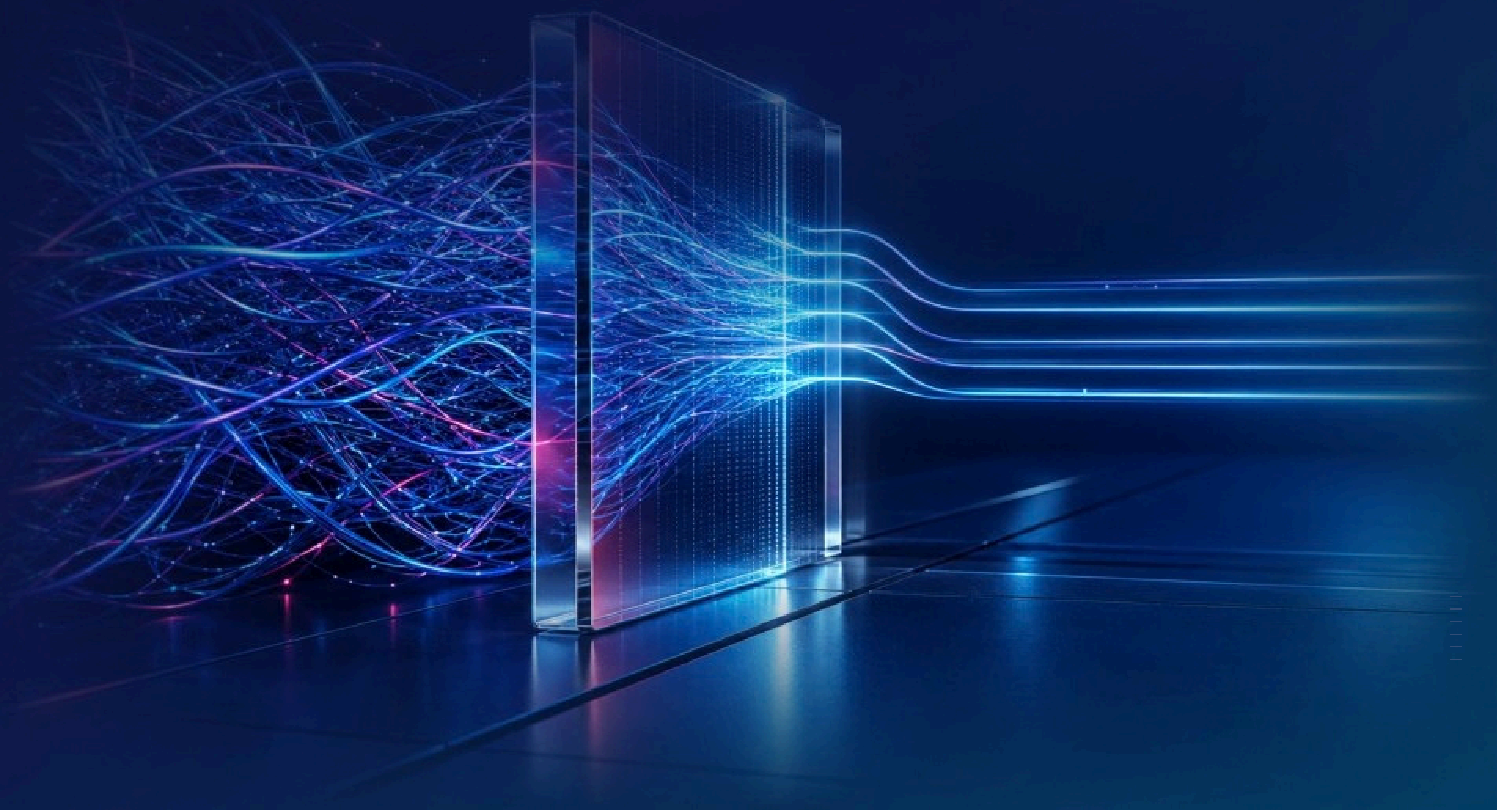
This creates room for architects to spend less time producing first drafts and more time examining trade-offs: resilience versus cost, reuse versus isolation, immediacy versus long-term maintainability. Yet the quality of the result depends on whether the agent can access the bank's target architecture, non-functional requirements, technology standards and previous architecture decisions. A plausible design is not necessarily an admissible design.

1.3 Development: AI is fast only when context is explicit

Development is where the benefits of AI are most visible. Coding assistants can complete functions, translate between programming languages, explain legacy code and draft documentation. More autonomous agents can decompose features, modify multiple files, run tests, analyze failures and submit changes for review. In a bounded experimental task, AI assistance reduced completion time by 55.8%, illustrating the potential for substantial gains where the task and context are well defined.

The gains are not universal. A 2025 randomized trial involving 16 experienced open-source developers and 246 tasks found that developers working in large, familiar repositories took 19% longer when using AI tools, even though they believed AI had made them faster. The researchers identified repository complexity and implicit project knowledge as contributing factors. The contrast between these studies matters: AI performs best when the task is bounded and the required context is explicit; its advantage can erode when critical knowledge remains tacit, dispersed or difficult to retrieve.

Reconstructing tacit knowledge changes that equation. For a critical engagement we found that, migrating roughly 1,000 mission-critical trading and client-facing screens off a proprietary low-code platform, hampered by scarce legacy skills and undocumented tribal knowledge, would ordinarily take a per screen effort of about a week. Reconstructing application intent, rather than translating code line by line, cut this effort to under a day (more than 80% reduction). In terms of specialist engineer weeks, reduction happened from about 1,000 specialist engineer-weeks to 500–700 engineer-weeks.





1.4 Quality: generated tests can certify the wrong system

AI can generate unit, system, integration and regression tests; derive scenarios from acceptance criteria; create synthetic test data; and identify unusual execution paths. Test agents can map code changes to impacted functions, prioritize test suites and repair scripts when interfaces change. This moves quality engineering earlier in the lifecycle and creates shorter feedback loops between code generation and validation.

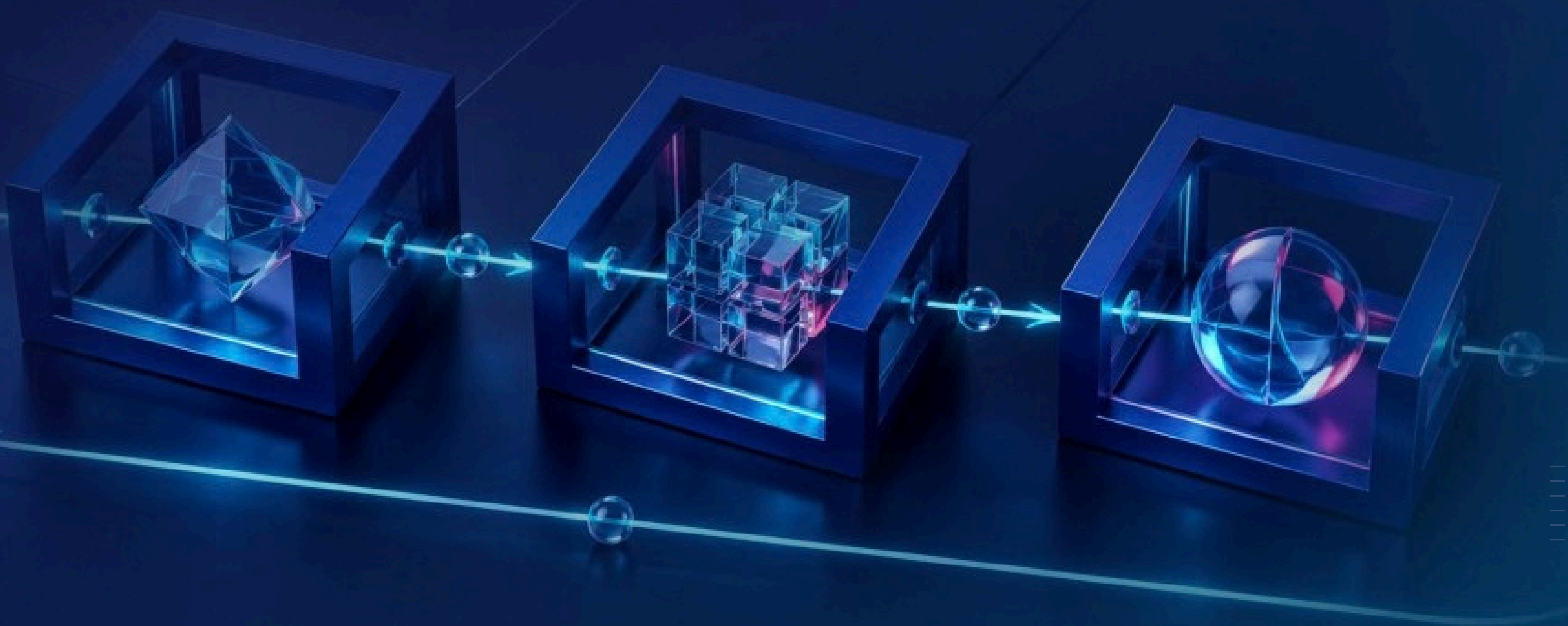
However, functional output should not be mistaken for secure output. Veracode's 2025 assessment of more than 100 large language models found that AI-generated code introduced security vulnerabilities in 45% of its security-relevant tests. That evidence reinforces a practical rule: every generated artefact must remain subject to secure coding standards, automated scanning, traceable test evidence and risk-based human review.



1.5 Production: operations is the next requirements document

In production, AI agents can correlate telemetry across applications and infrastructure, detect anomalies, isolate likely causes and recommend remediation. Within controlled boundaries, they can scale resources, restart services or trigger rollbacks. Operational incidents then become new context: defect patterns, performance behavior and customer impact can inform subsequent requirements, designs and tests.

This closes a loop that traditional SDLC models often leave open. Production is no longer the end of delivery; it is the richest source of evidence for what the delivery system should learn next.

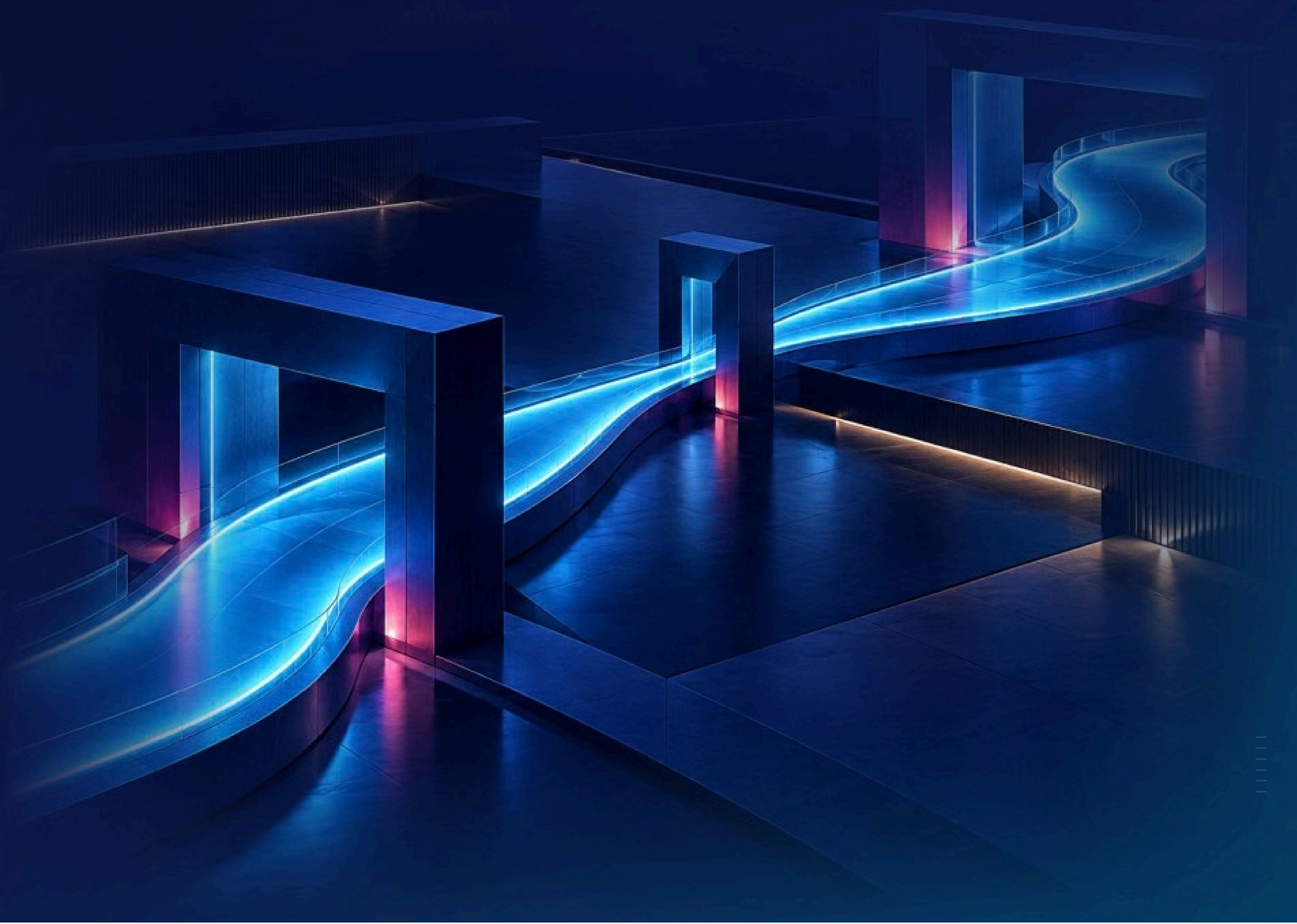


1.6 Activity is not flow

Individual productivity does not automatically translate into delivery productivity. A developer may generate code in minutes, but the feature will not reach production faster if test environments, security approvals or deployment pipelines remain constrained. Optimizing one stage can simply move the queue downstream.

Google DORA's evidence captures this distinction. In 2024, increased AI adoption correlated with a 7.5% improvement in documentation quality, 3.4% improvement in code quality and 3.1% improvement in code-review speed, but also with lower delivery stability. By 2025, Google DORA found a more positive relationship with throughput and product performance, while delivery stability remained under pressure, especially where automated testing, version control and feedback loops were weak.

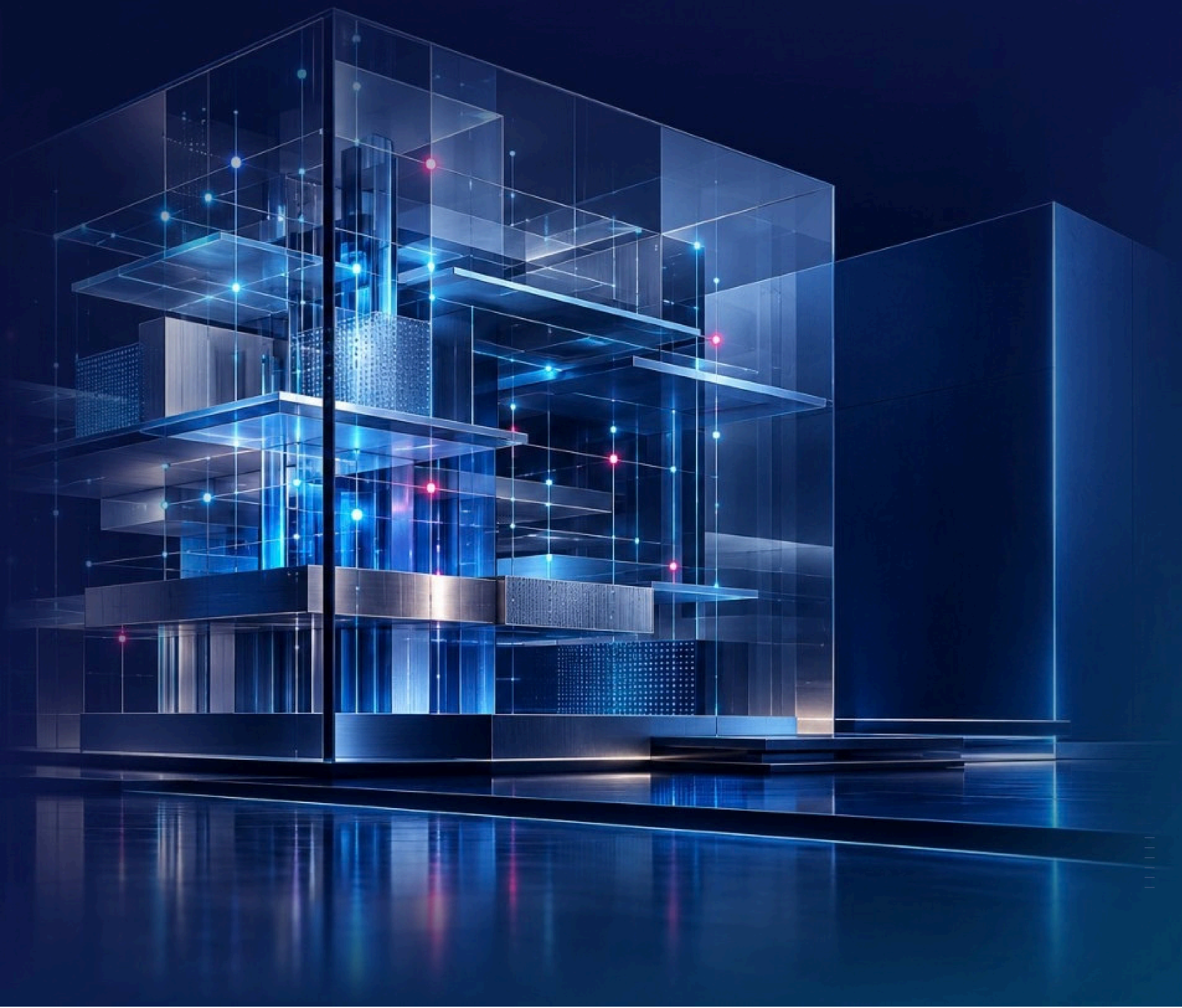
Multi-agent orchestration can connect work across stages, coordinate hand-offs and invoke human approval at defined gates. Yet orchestration alone is not enough. A digital assembly line still produces the wrong outcome if every station receives a different specification. End-to-end acceleration depends on an unbroken context continuum.



2. The bank the agent cannot see

Context engineering is the discipline of curating and maintaining the information an AI system needs to produce a desired outcome. Anthropic describes the challenge as selecting the most useful information from an evolving universe of possibilities within the model's practical limits. In an enterprise SDLC, that definition must extend beyond tokens. Context is a governed system of intent, knowledge, constraints, evidence and memory made available to an agent at the moment it reasons or acts.

That system determines whether an agent understands not only what to do, but why, within which boundaries, using which evidence and with whose authority. It carries the logic of the organization from demand to production.



2.1 The banking context stack

Context type	What it carries	Example in banking
Intent context	Customer need, business outcome, scope, priority and success measures	Reduce legitimate card declines without weakening fraud controls
Domain context	Product rules, transaction states, thresholds, taxonomies and exceptions	Eligibility and cooling-off rules for a credit-limit increase
Customer context	Personas, journeys, consent, channel preferences and accessibility needs	Assisted journeys for vulnerable or digitally excluded customers
Regulatory context	Obligations, jurisdictions, evidence requirements and retention rules	Fair-lending, KYC, privacy and model-risk obligations
Architecture context	Target state, approved patterns, dependencies and decision records	Event-driven pattern, approved cloud services and core-banking constraints
Engineering context	APIs, schemas, coding conventions, libraries and repository knowledge	Authorized customer API, error-handling standard and dependency versions
Quality context	Acceptance criteria, coverage, defects and release thresholds	Required negative scenarios and zero tolerance for severity-one defects
Operational context	Telemetry, service levels, incidents, capacity and runbooks	Peak-payment load profile and rollback procedure
Security context	Threat models, data classification, access entitlements and control policies	Mask PII, restrict secrets and require approval for privileged actions
Provenance and trust context	Source, version, owner, confidence, lineage and expiry	The policy version used to generate and approve a rule

The context required by a requirements agent is not the same as that required by a test or release agent. The objective is therefore not to load everything. Anthropic notes that recall and reasoning can degrade as irrelevant material accumulates, a phenomenon it calls “context rot”. Microsoft likewise identifies relevance, redundancy reduction, memory, hybrid retrieval and intelligent compression as central to efficient and predictable performance.

This leads to a core design principle: provide the smallest set of high-signal, authorized and current information needed for the decision at hand. Quality outranks quantity. Traceability outranks convenience. An agent should be able to identify where its context came from, which version it used, what may be missing and how confident it is in the result.

3. Context loss across stages

Every SDLC hand-off is a compression event. A customer need becomes a requirement; the requirement becomes a design; the design becomes code; the code becomes test evidence; and the release becomes operational behavior. At each transition, detail is summarized, translated or dropped. In a human-led process, experienced practitioners often fill the gaps through conversation and institutional memory. AI agents cannot reliably infer what the organization has failed to make explicit.

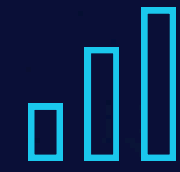
Context is lost when a downstream agent receives the instruction but not the rationale. It may know what feature to build, yet not which outcome it supports, which exception rules apply, why an architectural decision was taken, what residual risk remains or which data version is authoritative. Each stage then operates within its own internally consistent but mutually incompatible version of reality.



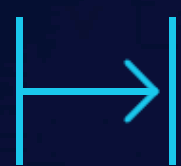
3.1 Eight ways meaning disappears



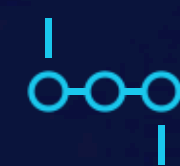
1. **Ageing:** A policy, schema or dependency changes, but the retrieved context still reflects an older state.



2. **Omission:** Assumptions, exceptions or non-functional requirements never enter the formal artefact.



3. **Substitution:** An agent fills a gap with general model knowledge that sounds reasonable but is wrong for the enterprise.



4. **Translation:** Meaning changes as business language becomes technical specifications, code and tests.



5. **Fragmentation:** A complete decision is split across tickets, documents, chats, repositories and approval tools.



6. **Access failure:** The right context exists, but permissions, indexing or integration prevent the agent from reaching it.



7. **Compression loss:** Summarization removes an exception or rationale that later becomes critical.



8. **Contamination:** Untrusted or malicious content enters prompts, memory or retrieval stores and influences subsequent action.

These are not abstract risks. In one modernization program, we found that undocumented tribal knowledge across roughly 1,000 legacy areas was the single largest driver of migration cost and risk - the same omission and fragmentation patterns mentioned above, at enterprise scale.

3.2 One journey: credit limit increase

A development agent can quickly create the interface and rule execution logic for credit limit increase. But if it does not receive customer eligibility criteria, affordability requirements, fraud checks, channel rules, jurisdictional constraints and adverse-action obligations, it can generate code that works technically while failing operationally or regulatorily. If the test agent then derives cases from the same incomplete story, the error becomes more profound.

Stage	Context the agent must receive	Who signs the contract	What fails if it is missing
Requirements	Intent, domain, customer, regulatory	Product, with risk	Story ships without affordability, cooling-off or adverse-action obligations
Design	Architecture, security, regulatory	Enterprise architecture, with CISO	A plausible pattern that violates core constraints or data-residency rules
Development	Engineering, domain, security	Engineering	Code that executes the wrong threshold, in the wrong jurisdiction
Quality	Quality, domain, regulatory	Quality engineering, with risk	Tests certify the incomplete story and create false assurance
Release / operations	Operational, security, provenance	Operations, with control	No rollback path, no policy version, no audit lineage

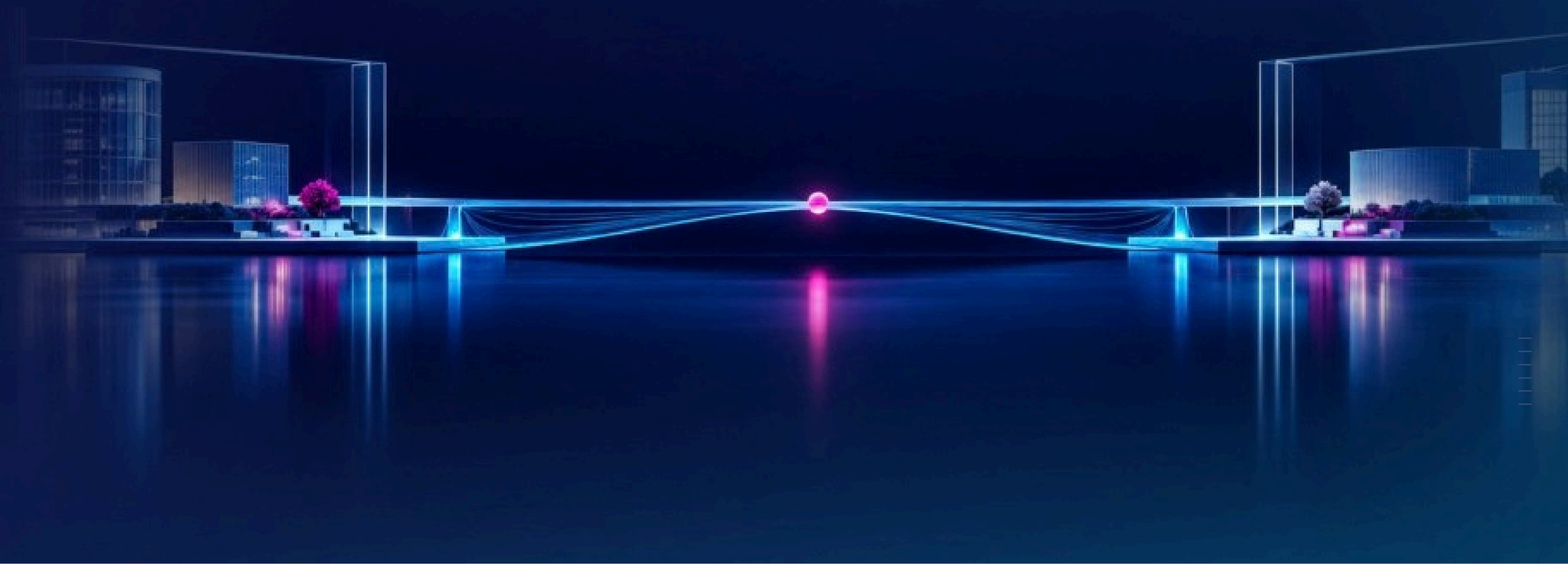


The effects compound across the lifecycle: requirement drift, inconsistent design, insecure code, incomplete testing, release delay and weak auditability. More output also increases the review burden. The 2025 Stack Overflow survey found that more developers distrusted AI accuracy than trusted it, and that experienced practitioners were particularly cautious. In banking, this trust gap can become a control gap when reviewers cannot see the sources, assumptions or decision path behind an AI-generated artefact.

Context also expands the security perimeter. IBM's 2025 breach research found that 13% of surveyed organizations had experienced a breach involving AI models or applications; among those organizations, 97% lacked proper AI access controls. A context continuum must therefore be protected like any other critical enterprise system. Sensitive data should be minimized, access should follow least privilege, agent actions should be bounded and every consequential decision should be logged with provenance.

This risk is sharper for tier two and tier three banks. Most regional and community banks consume AI capability through core-banking vendors, fintech partners and managed-service providers. That means the context an agent needs often sits outside the bank's own walls: in a vendor's data model, a partner's API, or a core system (Fiserv, FIS, Jack Henry and similar platforms still run the majority of US community-bank cores) that the bank does not control and cannot easily instrument.

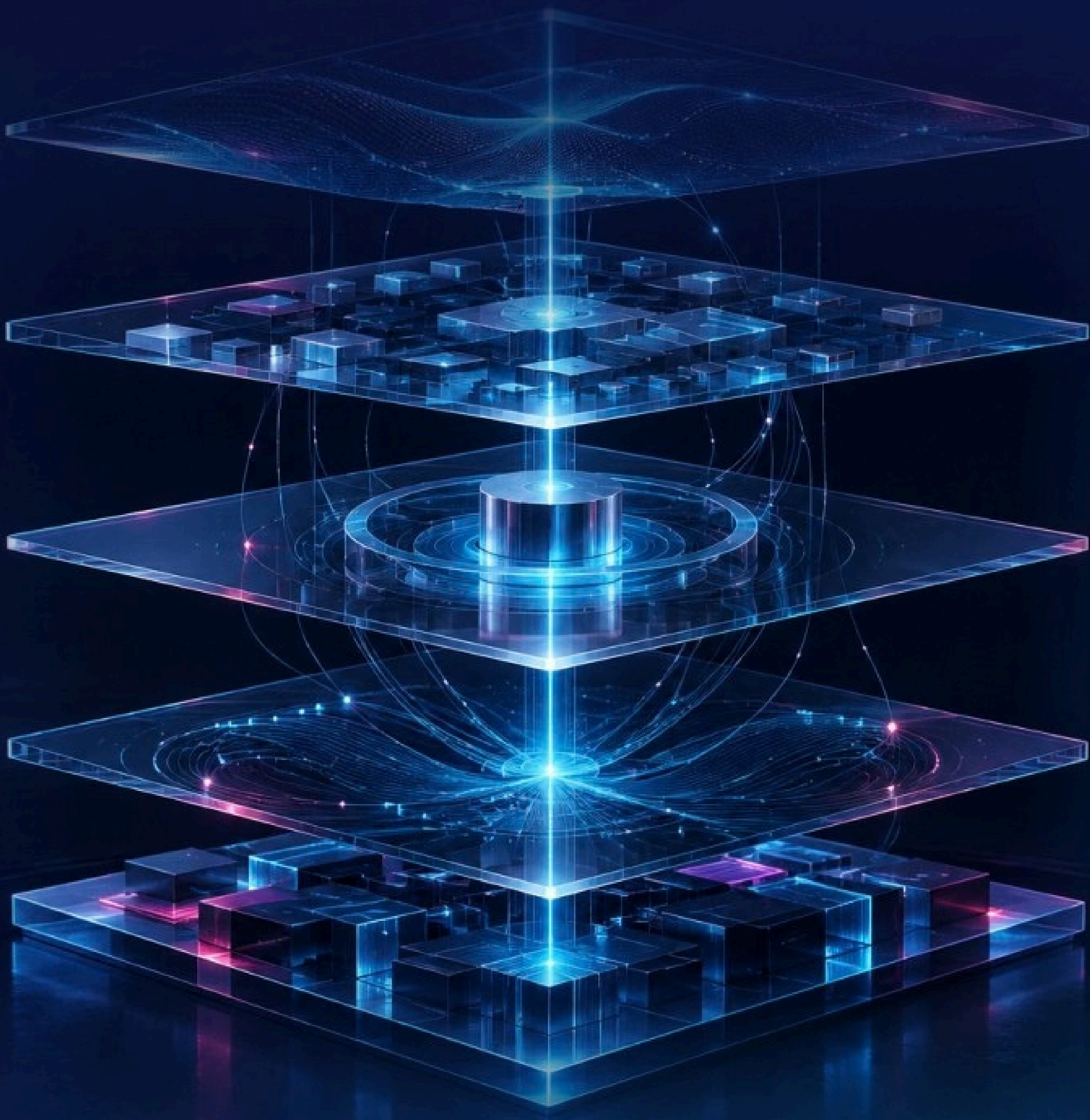
Third-party risk management is therefore not adjacent to context engineering for this segment — it is part of it. Interagency guidance on third-party relationships already requires banks to understand how a vendor's systems make decisions that touch customers; AI agents acting inside or alongside those systems raise the same question with less visibility. A context contract that stops at the bank's own boundary is incomplete. For tier 2/3 institutions, context engineering must extend into vendor and partner integration points, not just internal delivery teams.



4. Engineer a context operating system, not another layer

The answer is not a single “context layer”. A layer suggests a passive place from which information is fetched. AI-powered delivery requires something more dynamic and deep: a banking specific context operating system that continuously captures domain knowledge, assembles task-specific context, governs agent action and learns from outcomes.

Google DORA’s 2025 guidance explicitly recommends connecting AI to internal context and investing in internal platforms, foundational practices and safety nets. NIST’s Generative AI Profile similarly emphasizes governance, measurement, monitoring and provenance across AI systems. Together, these point to an operating model in which context is engineered as a controlled enterprise asset.



4.1 Context contracts: a verifiable transfer of meaning

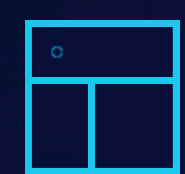
At the end of each stage, the organization should create a context contract, a machine-readable, human-reviewable record that defines:



- Required inputs and authoritative sources



- Generated outputs and linked artefacts



- Decisions, assumptions and rationale



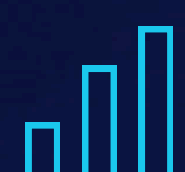
- Constraints, policies and control evidence



- Owner, approver and accountable role



- Confidence, unresolved questions and residual risk



- Permitted actions and escalation conditions



- Version, lineage, timestamp and expiry criteria

A context contract turns a hand-off into a verifiable transfer of meaning. The design agent can show which requirements and standards shaped an architecture; the coding agent can show which design decision and API version it used; the test agent can connect every critical scenario to acceptance criteria and controls. Traceability becomes part of execution rather than documentation assembled after the event.

In practice, agents log confidence and rationale for every ambiguous judgment, so a reviewer can audit not just the output but the reasoning behind it. Dual control still applies to high-impact change: the checker must see provenance, confidence and residual risk, not only the generated artefact.

4.2 Worked contract: requirements → design, credit-limit increase

Field	Content
Journey / change	Credit-limit increase, unsecured revolving credit, UK and UAE books
Stage hand-off	Requirements → design
Intent	Reduce legitimate declines on existing prime customers without weakening fraud or affordability controls
Authoritative sources	Credit policy v4.2 (Risk); product rules v12 (Product); adverse-action standard (Compliance); customer-consent register (CX)
Required context types	Intent, domain, customer, regulatory, security
Outputs expected from design	Target architecture options, data-lineage view, API impact, control mapping
Constraints	No change to fraud-score cut-off without Financial Crime sign-off; PII masked in lower environments; UAE data residency retained
Assumptions	Cooling-off period remains 14 days; existing limit-increase caps still apply by segment
Unresolved / residual risk	Partner-hosted core eligibility flag not yet exposed as an API; treat as third-party context gap
Confidence	Medium. Policy versions current; core flag not instrumented
Owner / maker	Head of Cards Product
Approver / checker	Credit Risk, with Compliance on adverse-action wording
Permitted next actions	Design agent may propose patterns against the approved architecture catalogue; may not generate customer-copy or decisioning code
Escalation	Any change to eligibility, affordability or adverse action returns to Risk before design proceeds
Version / expiry	Contract v1.0; expires if credit policy or core eligibility schema changes

That is the difference between a story in a backlog and a hand-off a bank can defend.

4.3 Ten operating capabilities



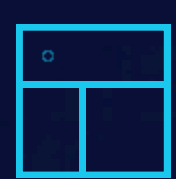
1. **Integration with existing SDLC processes and systems:** Connect product backlogs, document stores, architecture repositories, source control, CI/CD, test platforms, observability tools and policy systems. Context should meet agents within the workflow rather than requiring teams to duplicate knowledge.



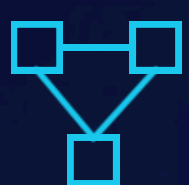
2. **Structured knowledge model:** Represent relationships among business outcomes, domain rules, decisions, services, APIs, data, controls, tests and incidents. This creates a navigable model of why the system exists and how it behaves.



3. **Semantic retrieval:** Retrieve by meaning as well as keywords, while respecting source authority, recency, sensitivity and the user's permissions.



4. **Stage-specific assembly:** Build a bounded context package for the task at execution time. Include what is relevant; exclude what is redundant, restricted or stale.



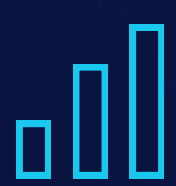
5. **Multi-model and agent orchestration:** Route work to the model or agent suited to the task, manage dependencies and preserve state across hand-offs.



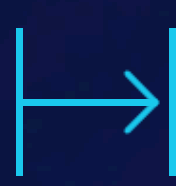
6. **Configurable, risk-based quality gates:** Apply stronger evidence, testing and human review to high-impact changes while allowing lower-risk work to flow faster — for example, pairing deterministic transformation rules with AI judgment reserved only for ambiguous exceptions.



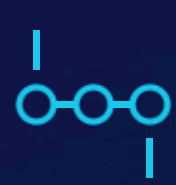
7. **Versioning and provenance:** Record the source, owner, version and transformation history of every context element and generated artefact.



8. **Evaluation and observability:** Measure groundedness of generated artefacts, retrieval quality, policy compliance, action success, latency, cost, defects and downstream rework—not simply lines of code generated.



9. **Feedback loops:** Feed production incidents, customer outcomes and control findings back into requirements, knowledge and tests.



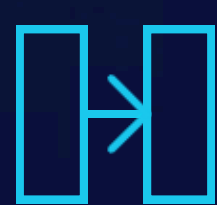
10. **Policy enforcement and access control:** Use least privilege, data minimization, segregation of duties, secret protection and explicit approval for consequential actions.

4.4 Five fabrics of Context operating system

These fabrics are the runtime of the capabilities above, not a second list to implement in parallel.



Knowledge fabric. This is the living digital twin of the domain. It goes beyond a document store or search index by connecting products, policies, processes, architecture, data, controls, code, tests and operational events. Its purpose is not to centralize every artefact, but to make relationships and authority discoverable.

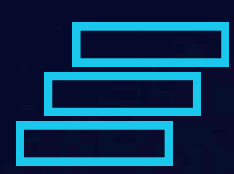


Integration fabric. This connects the knowledge fabric to the delivery toolchain. It captures changes and events, invokes tools through controlled interfaces, and carries context contracts between agents and human teams. Open interfaces matter, but so do identity, need for permission and isolation. An agent should never inherit more access than its task requires.



Governance and trust fabric. This defines what agents may access, infer and execute. It manages policy, risk classification, approval thresholds, provenance, audit logs and exception handling. NIST recommends tracking the origin and modification history of data and generated content to strengthen integrity and accountability.





Context assembly and memory. This selects current, relevant and authorized information for each action, then preserves durable learning outside the model's temporary window. Retrieval should combine structured relationships with semantic search, while compression should retain decisions and exceptions. Microsoft research has shown that compression can reduce long-horizon memory usage by 26–54% while preserving task performance, demonstrating that disciplined context reduction can improve efficiency without simply discarding meaning.



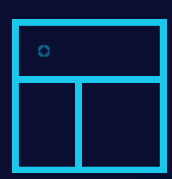
Evaluation and observability. Context quality must be measurable. Useful indicators include retrieval precision, stale-context rate, unsupported-claim rate, traceability coverage, policy violations, human override rate, rework, escaped defects and change-failure rate. The unit of success is not agent activity; it is a safe production outcome connected to business value.

Maveric built Context.OSAI to put this discipline into practice — connecting orchestration, closed-loop feedback, a contextual knowledge graph, agentic automation and safe-AI-by-design into a single operating layer for the AI-powered SDLC, as we've done for a top-tier bank's mission-critical trading estate.

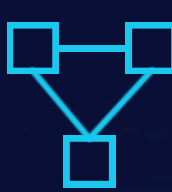


5. Action plan: Building the context continuum

A pragmatic adoption path starts with one material value stream rather than an enterprise-wide platform program.



1. **Choose a bounded journey.** Select a banking capability with meaningful change volume, visible hand-offs and measurable outcomes, for example, card-limit management, customer onboarding or payment exception handling. For banks that rely on vendor-hosted cores or fintech partners for much of the stack, choose a journey where at least one hand-off crosses that vendor boundary — for example, a credit-limit change that touches both an internal origination system and a partner-hosted core. This surfaces the third-party context gap early, rather than after the model has scaled.



2. **Map the context chain.** Trace how intent moves from demand through requirements, design, code, tests, release and production. Identify where rationale becomes inaccessible, stale or ambiguous.



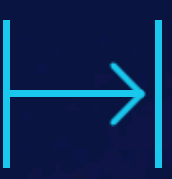
3. **Classify and assign ownership.** Group context into intent, domain, regulatory, architecture, engineering, quality, operations, security and provenance. Name an owner and authoritative source for each class.



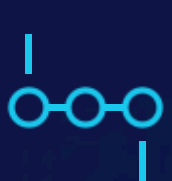
4. **Introduce context contracts.** Begin at the highest-friction hand-offs. Make assumptions, evidence, versions, confidence and permitted actions explicit.



5. **Connect the toolchain.** Integrate the existing systems of record before creating another repository. Preserve access controls and audit trails across retrieval and action.



6. **Engineer retrieval and assembly.** Use task-specific context packages, relevance ranking, freshness checks and compression. Test what happens when context is missing, conflicting or malicious.



7. **Apply progressive autonomy.** Start agents with read and recommend permissions. Expand to create, test and act only after evidence meets defined thresholds; retain human approval for high-impact changes, where engineers verify every AI-generated output before it is trusted, is one working version of this which we implemented in a critical engagement.



8. **Measure flow and trust.** Track lead time, rework, escaped defects, stability, context freshness, traceability and policy compliance. Compare AI-assisted value streams against a baseline.



9. **Close the production loop.** Convert incidents, overrides and customer outcomes into updated knowledge, controls and regression tests.



10. **Scale by pattern.** Reuse the operating model, schemas and controls across value streams while allowing each domain to retain its own rules and risk boundaries.

6. Conclusion: Context is the control plane for AI-native delivery

AI will continue to make individual SDLC activities faster. The strategic question is whether those gains accumulate into dependable delivery or dissipate into review effort, instability and risk. The answer will depend less on access to the latest model and more on the enterprise's ability to make its intent, knowledge and controls usable by machines.

Context engineering creates that ability. It turns fragmented information into an active, governed continuum; connects business intent to production evidence; and gives every agent the boundaries required to act responsibly. It also makes human oversight more effective—not by inserting approval everywhere, but by giving reviewers the provenance, confidence and residual risk needed to focus attention where it matters.

For banking CIOs, CROs, CISOs, and the directors who own product, architecture, engineering, quality and operations, the mandate is the same: move beyond deploying copilots and start engineering the context in which every copilot and agent operates. In an AI-powered SDLC, trust is not a final checkpoint. It is carried, tested and renewed at every stage.



References

1. McKinsey & Company, Unlocking the value of AI in software development, November 3, 2025: <https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/unlocking-the-value-of-ai-in-software-development>
2. DX, AI-assisted engineering: Q1 impact report, 2026: <https://getdx.com/uploads/ai-assisted-engineering-q1-impact-report.pdf>
3. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot: <https://arxiv.org/abs/2302.06590>
4. <https://cloud.google.com/blog/products/devops-sre/announcing-the-2024-dora-report>
5. <https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report>
6. <https://stackoverflow.co/internal/resources/2025-stack-overflow-developer-survey-for-leaders/ai-adoption/>
7. <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>
8. <https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/>
9. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>
10. <https://www.microsoft.com/en-us/research/project/efficient-ai-applications-context-engineering-and-agents/>
11. <https://newsroom.ibm.com/2025-07-30-ibm-report-13-of-organizations-reported-breaches-of-ai-models-or-applications,-97-of-which-reported-lacking-proper-ai-access-controls>
12. <https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf>
13. <https://www.microsoft.com/en-us/research/publication/acon-optimizing-context-compression-for-long-horizon-llm-agents/>