

WHITEPAPER · AI ENGINEERING

The AI SDLC in the Enterprise

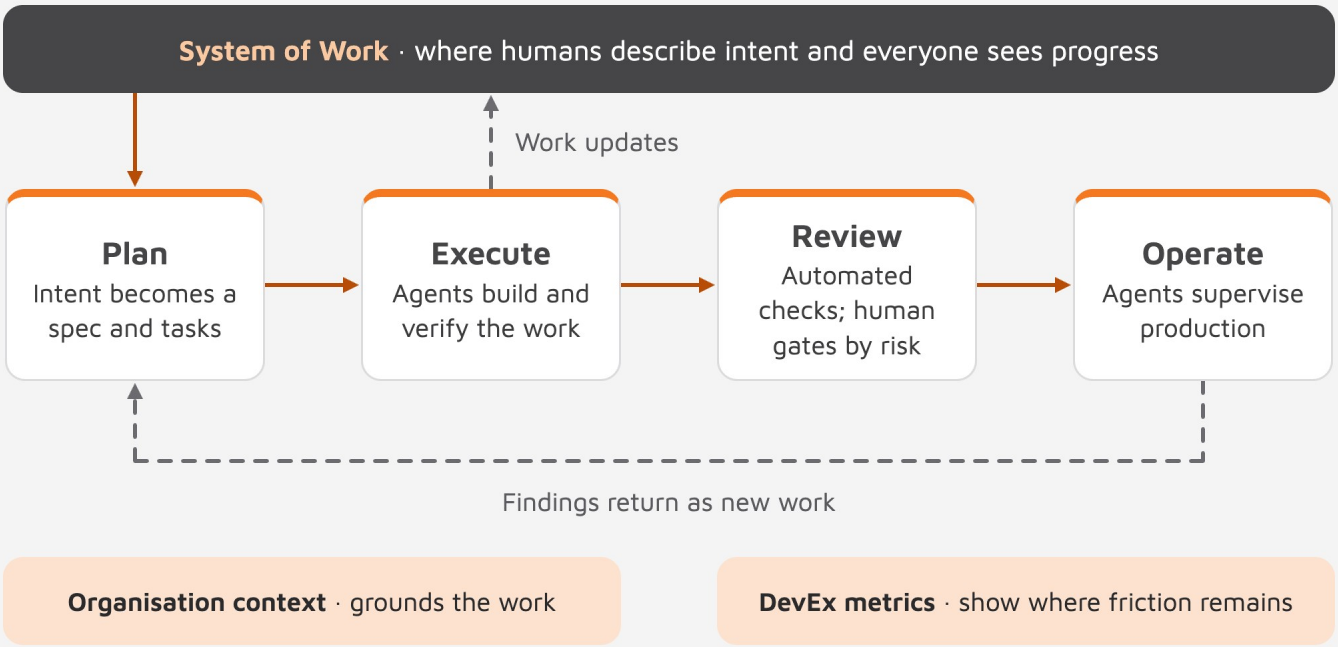
How leaders can set one clear direction for agentic software delivery - and the engineering building blocks it depends on.

By Michael Leroy, Principal Consultant

EXECUTIVE SUMMARY

A model for agentic software delivery

AI now writes code faster than most organisations can plan, review and release it. The AI SDLC is a model for turning that speed into delivery capacity, safely: **humans describe what needs to be built, agents take it from there, and people step in where the risk requires it.** Organisational knowledge grounds every decision, and DevEx metrics show where friction remains, while supporting incremental adoption of agentic capabilities.



A simplified view. The full framework is set out on page 5.

What it gives the enterprise

Delivery capacity, not just faster coding

Agents do the implementation, so your people spend their time on intent, judgement and risk, and work moves through review and release at the pace it is written.

The business shapes the work directly

Through the System of Work, stakeholders and analysts put intent straight into the delivery pipeline and see progress as it happens.

Governed by design

Risk-tiered human gates and a complete audit trail make agentic delivery safe for regulated enterprises.

Every team working at its best

Shared context, skills and guardrails let every team deliver to the standard of your strongest, rather than maturity pooling in a few.

Key takeaways

Agentic delivery is here, and organisations that get it right are already seeing the gains. These are the points every engineering leader should carry into the conversation with their board and their teams.

AI IS PRODUCING MORE. THE SYSTEM AROUND IT DECIDES WHETHER YOU KEEP IT.

+75.5%

growth in weekly PR throughput over four quarters across 500+ organisations: **output is up**

105%

longer wait for a first review where AI code generation is highest: **the gain stalls in the queue**

Weeks to days

at OpenAI, where the whole lifecycle was rebuilt around agents: **what happens when you fix the system**

Sources: DX, State of AI Impact in Engineering, Q3 2026; OpenAI, Building an AI-native engineering team.

01 **Adoption has outrun strategy.** AI is everywhere in engineering, but few organisations have set a North Star for how it changes the SDLC - so silos form and maturity becomes uneven.

02 **The bottleneck has moved.** Writing code is no longer the constraint. Planning, review and deployment still run at human speed, and that is where the gains are lost.

03 **Engineering building blocks are non-negotiable.** CI/CD, automated testing, static analysis and a DevOps mindset are prerequisites for AI adoption. AI amplifies the bad just as much as the good.

04 **Enterprises need a System of Work.** Frontier-lab playbooks are developer-centric. In an enterprise, business stakeholders, analysts and architects shape the work too, and they need a place to collaborate.

05 **Humans move to intent, judgement and risk.** Humans describe what needs to be built; agents take it from there. A risk profile, not a blanket rule, decides where a human stays in the loop.

06 **Context as an engineering discipline.** An agent is only as good as the organisational knowledge it can reach. Treat context as an engineering asset that is designed, owned and maintained.

07 **Measure, then adopt incrementally.** Baseline with DX Core 4, never treat AI usage as a productivity signal, and adopt agentic capabilities one at a time as confidence grows.

WHY NOW

Everyone has adopted AI. Few have a strategy for it.

23%

of leaders say their organisation has a clearly communicated, actively implemented AI strategy across the SDLC (17% of developers)

49%

of developers spend four or more hours a week reviewing AI output, correcting prompts and more

75%

of developers say they now carry more responsibility for validating the quality and safety of AI output

Source: Atlassian State of Developer Experience Report 2026.

Nearly every engineering organisation has adopted AI, yet few have a clearly communicated, actively implemented strategy for it across the software development lifecycle (SDLC).

This tells us that things are moving so fast that most organisations haven't yet had the chance to step back and think through the implications end to end, beyond pushing for AI adoption. In the meantime, engineers are experimenting, and that should be encouraged, but not at the expense of a North Star. Without one, AI silos appear, maturity with AI tooling becomes uneven, and some teams are left behind for want of clear guidance on what AI means for the traditional SDLC.

To make matters worse, a tidal wave of announcements and new approaches lands every week, and filtering the valuable from the noise is increasingly difficult. Leaders aren't sure which developments deserve their attention, let alone which to evaluate for adoption in their organisation.

> **How can leaders communicate a clear vision for an AI SDLC, so that delivery and innovation are encouraged while every team steers in the same direction?**

→ We believe the answer lies in understanding how the SDLC evolves with agentic delivery, and in a clear framework that highlights the building blocks that must be in place for AI to be applied. **That framework is the AI SDLC set out in this paper.**



What the frontier is telling us

Frontier labs now publish how they build software with their own agents. Two accounts stand out.

ANTHROPIC · AUGUST 2026

The AI-native SDLC playbook

> "Code is no longer the bottleneck."

- Six stages - Plan, Design, Build, Test, Deploy, Maintain - run as a loop, not a line.
- Each stage commits an artefact the next one reads - intent, spec, plan, diff, review - which doubles as the audit trail.
- Governance is enforced as the agent acts: skills encode policy and hooks act as approval gates.

OPENAI · SEPTEMBER 2026

The agentic software factory

> "Handhold this change until it is safely and fully rolled out into production."

- A human defines the outcome; the agent implements and babysits CI until the change is green.
- **Changes are classified by risk**: stricter paths for high risk; auto-approval for low risk in opted-in areas.
- An agent shepherds each approved change into production.

Building on it for the enterprise

Both are excellent starting points, and we build on top of them. But the approach is developer-centric. In an enterprise, work takes shape collaboratively - business stakeholders, business analysts and architects all shape what gets built - and needs a **System of Work** designed to support that collaboration. Agents are excellent with Markdown files; they are far less effective at promoting feedback loops and cross-team collaboration, especially for non-technical stakeholders. From OpenAI, the standout idea is **risk profiling** each change to decide whether a human belongs in the loop.

	DEVELOPER CENTRIC	ENTERPRISE REALITY
Who shapes intent	Engineers and product owners, in the repo	Business stakeholders, BAs, architects, risk and change teams
Where work lives	Markdown artefacts in version control	A System of Work (tickets, docs, approvals), linked to the code
Human gates	Risk-tiered review; low risk auto-approved	The same, plus change boards and regulatory traceability

THE FRAMEWORK

The AI SDLC: agentic software delivery

Our framework describes an AI-assisted SDLC where **humans describe what needs to be built and an agent takes it from there.** Humans are placed in the loop where the risk requires it. Organisational knowledge is made available through a context layer, so every decision is grounded in it. And the performance of the AI SDLC is measured through DevEx metrics to highlight friction points.

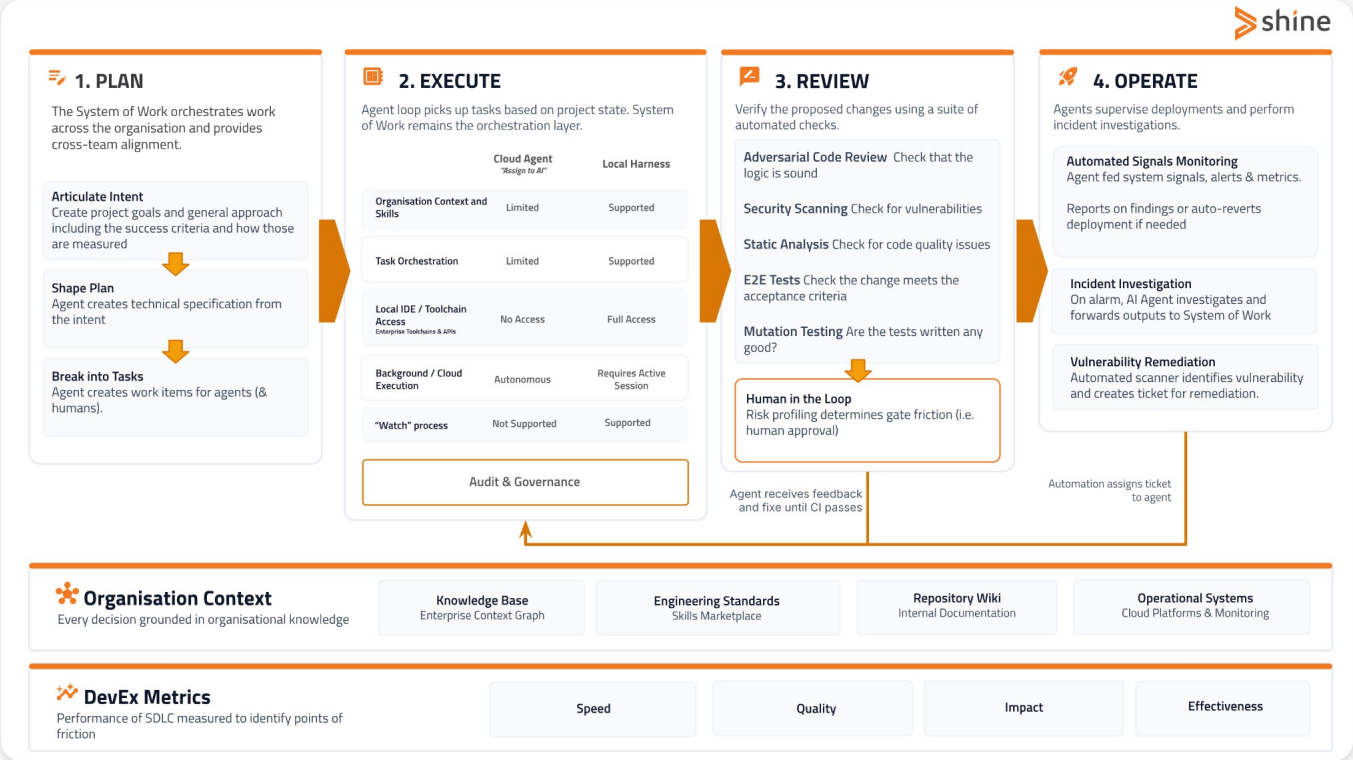


Figure 1. The AI SDLC: four stages joined by feedback loops, grounded in context, measured by DevEx.

Four stages, one loop

Plan, Execute, Review and Operate. Review feedback and production findings flow back to the agent as new work.

The System of Work

The orchestration layer where intent is shaped, work is assigned and progress is visible to everyone, technical or not.

Organisation Context

Knowledge base, engineering standards, repository wiki and operational systems, feeding every stage.

DevEx Metrics

Speed, quality, impact and effectiveness, measured across the whole SDLC to locate the points of friction.

Stage 1 - Plan

The System of Work orchestrates work across the organisation and keeps teams aligned. Humans articulate intent; agents turn it into a specification and tasks.

HUMAN-LED
Articulate intent
Goals, general approach, success criteria and how they're measured

AGENT-LED, REVIEWED
Shape plan
A technical spec grounded in the codebase, architecture and standards

AGENT-LED
Break into tasks
Work items for agents (and humans) in the System of Work

How coding agents help

- Interview the originator as an analyst would - scope, users, constraints, success criteria - and capture intent in their own words.
- Cross-reference intent against the codebase to flag ambiguity, trace affected services and surface dependencies.
- Apply security, compliance and UX standards while the spec is written, not weeks later in review.
- Break the spec into right-sized & linked work items and write them back to the System of Work.

What humans do now

BEFORE Requirements distilled by committee, written up by hand
NOW Stakeholders articulate intent and how success is measured

BEFORE Analysts and architects hand-write the specification
NOW Humans review the agent's spec and resolve the concerns it flags

BEFORE Tickets sized and split in refinement meetings
NOW Humans own priority, sequencing and trade-offs

Intent → spec → tickets: how far should specs go?

The flow draws on Anthropic's playbook and spec-driven development (SDD), which Thoughtworks' Birgitta Böckeler splits into three levels:

WE RECOMMEND
Spec-first
A spec is written first and drives the task at hand.

Spec-anchored
The spec is kept and evolved with the feature.

Spec-as-source
Humans only edit the spec; code is generated from it.

The spec is written to guide the agent's output and kept in the System of Work for context, but is no longer updated once that work is completed. English may be the hottest new programming language, but it isn't ready to be the source: the same sentence can be read differently by two people, or by one model twice. **The code remains the source of truth.**

Stage 2 - Execute

An agent loop picks up tasks based on project state. The System of Work remains the orchestration layer: what is ready, who picks it up, where progress shows.

How coding agents help

- Pick up ready work items, read the linked spec and implement end to end: code, tests and documentation.
- Verify their own work - builds, tests, linters - and iterate until the checks pass, before a human sees it.
- Run in parallel in isolated sandboxes, so one engineer can steer several streams of work.
- Report progress and pull requests back to the System of Work for all to see.

What humans do now

BEFORE Engineers translate specs into code, line by line

NOW Engineers steer agents and resolve the ambiguities they surface

BEFORE One engineer, one task at a time

NOW One engineer orchestrates several agent sessions

BEFORE Conventions live in people's heads

NOW Engineers encode them as shared context, skills and guardrails

Cloud agents and local harnesses

CAPABILITY	CLOUD AGENT	LOCAL HARNESS
Organisation context and skills	Limited	Supported
Task orchestration	Limited	Supported
Local IDE / enterprise toolchains and APIs	No access	Full access
Background / cloud execution	**Autonomous**	Requires an active session
"Watch" process	Not supported	Supported

Today the two have complementary strengths: cloud agents run autonomously but see less of the organisation; local harnesses see everything but need someone at the keyboard. **Match the execution environment to task complexity**: for now, reserve more ambitious work for local harnesses and let cloud agents pick up well-bounded, lower-complexity tasks. We expect the gap to close as the frontier labs invest in managed cloud sandbox environments, so design the System of Work to leverage either.

Audit & governance spans both. Every agent action is attributable to an identity, permissions are managed centrally, the chain of intent, spec, plan, diff and review forms the audit trail, and an AI gateway can apply policy, cost controls and logging to every model and tool call.

Stage 3 - Review

Proposed changes are verified by a suite of automated checks. The agent receives the feedback and fixes until CI passes; humans step in where the risk requires it.

A suite of automated checks

- ✓ **Adversarial code review** - a separate agent, with a fresh context, checks the logic is sound
- ✓ **Security scanning** - vulnerabilities, secrets and dependency risk
- ✓ **Static analysis** - code quality and maintainability issues
- ✓ **End-to-end tests** - the change meets the acceptance criteria in the spec
- ✓ **Mutation testing** - are the tests any good? Mutated code should make them fail

REVIEW IS THE NEW BOTTLENECK

49%

of developers spend 4+ hours a week reviewing AI output and correcting prompts

105%

longer wait for a first review where AI code generation is highest

Atlassian DevEx Report 2026; DX Q3 2026

Human in the loop: risk profiling determines gate friction

Low risk

Docs, copy, config and dependency bumps in well-tested areas.

Gate: automated checks and agent approval; humans sample.

Medium risk

Feature changes within a service's established patterns.

Gate: checks plus code-owner approval, focused on intent and design.

High risk

Auth, payments, personal data, schema migrations, infrastructure, regulated code.

Gate: specialist reviews, a named approver and change sign-off.

Illustrative tiers. Start conservatively and move a type of change down a tier as evidence accumulates.

What humans do now

BEFORE Humans read every line, and quality varies with reviewer load

NOW Every change gets the same automated baseline; humans judge intent, architecture and risk

BEFORE One approval rule for every change

NOW Humans own the review policy and the risk model that routes each change

Stage 4 - Operate

Agents supervise deployments and perform incident investigations. What they find flows back into the System of Work as new work, closing the loop.

Automated signals monitoring	The agent is fed system signals, alerts and metrics. It reports on findings, or auto-reverts a deployment if needed through a pre-approved rollback path.
Incident investigation	On alarm, an agent correlates logs, recent deployments and code changes, proposes likely causes and mitigations, and forwards its outputs to the System of Work.
Vulnerability remediation	Scheduled scanners identify vulnerabilities and create tickets for remediation, which automation then assigns to an agent.

CLOSING THE LOOP

Signal or finding in production ▸ **Ticket raised** in the System of Work ▸ **Assigned** to an agent ▸ **Back to Execute** and Review

Keep detection deterministic. Thresholds on metrics decide when an agent is invoked; a tier decides what it may do - log, diagnose, or propose a fix through the normal review gate. OpenAI's incident agent, Sevbot, gathers context and proposes mitigations in the incident channel, but applies one only when an engineer tells it to.

What humans do now

BEFORE · Correlate logs, commits and infrastructure changes by hand, under pressure

NOW · Validate the agent's root cause and approve the remediation

BEFORE · Post-incident actions languish in a backlog

NOW · Triage agent findings: fix now, schedule or dismiss

BEFORE · Security scans are point-in-time events before a release

NOW · Scans run on a schedule; humans review the proposed fixes

Humans keep authority over novel incidents and sensitive production changes. Agents act only through pre-approved routes: a rollback, a runbook or a pull request into the review gate.

ORGANISATION CONTEXT

Building the organisational context layer

Every agent decision should be grounded in organisational knowledge. In an enterprise, that knowledge is scattered across knowledge bases, standards, wikis, operational systems and people's heads. Getting it to the agent in the right shape is the work.

CAFE(S): WHAT GOOD CONTEXT LOOKS LIKE

"...even state-of-the-art models degrade rapidly when operating on unclear, incomplete, or stale inputs."

C	A	F	E	(S)
Clarity	**Actionability**	**Fidelity**	**Efficiency**	**Security**
Can the agent read the request as intended?	Clear goals, boundaries and a way to know it's done?	Is it true at the moment the agent reads it?	Scoped to the task, without wasted tokens?	Should the agent have this context at all?

Houck, Kanat-Alexander, Kalliamvakou, Storey & Forsgren - CAFE(S), ACM Queue / DX Research, 2026

From search to knowledge graphs and LLM wikis	Retrieval is moving beyond chunk-and-search. Graph RAG captures the relationships between systems, teams and decisions in an enterprise context graph. LLM wikis - a pattern popularised by Andrej Karpathy in April 2026 - have agents compile raw sources into interlinked Markdown, file answers back so knowledge compounds, and "lint" it for inconsistencies.
CLI or MCP? Both.	CLIs are token-efficient and well known to models, which makes them strong in a developer's inner loop. MCP earns its overhead in the outer loop, where agents cross system boundaries and access must be governed: credentials stay behind a scoped, auditable interface rather than being handed to every session.
Skills registry	Skills are an effective medium for sharing reusable process definitions and standards across the organisation. A central registry lets teams publish, discover and reuse them. As vendors converge on agent skills standards, those skills can be versioned and updated like code dependencies, so a change to a standard reaches every team's agents.

Treat context as a first-class engineering asset: give critical sources an owner and a review cadence, and review shared context files with the same rigour as code.



DEVEX METRICS

You can’t improve what you can’t measure

Make data-driven decisions wherever possible: it takes the emotion out of AI conversations with boards and teams alike. DX Core 4 remains an excellent framework for measuring engineering performance across four counterbalanced pillars.

Speed

How quickly do teams deliver value?

Key metric: diffs per engineer

Effectiveness

How well do processes support engineers?

Key metric: Developer Experience Score

Quality

How stable is what we ship?

Key metric: change failure rate

Impact

Is effort going to new capabilities?

Key metric: % of time on new capabilities

> **“Speed and throughput metrics, when used in isolation, often incite fear and counterproductive behaviors.”**
>
> DX Research, Measuring developer productivity with the DX Core 4

Measuring AI itself

DX’s AI Measurement Framework relies on three dimensions: **utilisation, impact and cost**. It recommends treating agents as extensions of the teams that direct them, because “every developer will increasingly operate as a ‘lead’ for a team of AI agents.” Booking.com used the approach across more than 3,500 engineers and saw a 16% increase in throughput within months.

USAGE IS NOT PRODUCTIVITY

64% of leaders say greater AI usage correlates with better outcomes

41% of developers use AI more than necessary because they feel expected to

Atlassian DevEx Report 2026

Measure outcomes, not activity - and start small. A monthly developer survey and the delivery data your pipelines already emit are enough for a baseline. Pair it with a context lens: DX’s CAFE(S) gives teams a shared vocabulary for reviewing context quality, though by design it is a definition rather than a measurement system.

WHAT YOU NEED IN PLACE

A strong engineering foundation. The price of admission for AI adoption.

The solution only runs on solid foundations. Organisations **must** invest in the engineering building blocks that let them adopt AI safely and reap the advertised efficiency gains. Point an agent at a codebase with thin test coverage, a manual release process and a quarterly security review, and it will generate more change than the organisation can safely absorb.

AI amplifies the bad just as much as the good.

> **"What hasn't kept pace is everything that makes that volume safe to trust."**
>
> DX, State of AI Impact in Engineering, Q3 2026

+75.5%

growth in weekly PR throughput over four quarters across 500+ organisations

CI/CD

Every change built, tested and deployed through an automated pipeline, with rollback rehearsed rather than improvised.

Automated testing

Unit, integration and end-to-end suites an agent can run itself: the feedback loop that lets it verify its own work.

Static analysis & security scanning

Deterministic quality and security gates on every change, running at the speed agents produce them.

A DevOps mindset

Teams own what they build through to production. Silos and hand-offs are removed.

There is very little excuse not to

AI is itself the opportunity to put these capabilities in place quickly. Modernisation efforts that previously didn't stack up financially are well worth revisiting now that agents have reduced the time to implement them. These are no longer nice-to-haves; they are prerequisites. Skip them and you'll be shipping slop fast, risking reputational damage to your brand.



ADOPTION

The AI SDLC journey - it doesn't have to be all or nothing

The agentic elements of the AI SDLC can be adopted gradually. Each step delivers value on its own and lays the foundations for the next, so teams can move at different speeds while still heading in the same direction.

Step 0
Foundations
CI/CD, automated tests, static analysis and a DevEx baseline

Step 1
Assisted
Local harnesses with shared context files and skills; humans prompt each step

Step 2
Delegated
Well-specified tickets assigned to cloud agents; automated review passes on every change

Step 3
Orchestrated
The System of Work dispatches agents from project state: intent → spec → tasks, with risk-tiered human gates

Step 4
Closed loop
Agents supervise deployments and investigate incidents; findings return to the System of Work as new work

Where to start

01
Follow the friction
Baseline with DX Core 4, then target the stage where friction is greatest - often not coding at all.

02
One stage, one team
Prove the step with a willing team, measure the outcome, then scale what works.

03
Codify and share
Turn what works into shared context, skills and guardrails, so maturity spreads across teams rather than pooling in one.

Bottom-up experimentation is how organisations learn what works. The framework is the North Star that keeps those experiments pointed the same way - so no team gets left behind.

CONCLUSION

The path forward: one direction, many teams

The AI SDLC isn't a tool purchase or a mandate from above. It is a direction: humans describe intent and manage risk, agents do the work, a System of Work keeps everyone aligned, context grounds every decision, and metrics show where friction remains.

Leaders who set that direction clearly - and invest in the engineering building blocks beneath it - give their teams permission to experiment while steering together. That is how AI adoption turns from scattered pockets of individual speed into sustained, organisation-wide improvement.

Like everyone in the industry, we're still working out what best practice looks like in this new world, so treat this paper as our current thinking. We expect it to evolve as the tools mature and as we learn alongside our clients.

Navigating this landscape is complex, and you don't have to do it alone. If you'd like help defining your organisation's AI SDLC, assessing your engineering building blocks or establishing a measurement baseline, get in touch with us at Shine Solutions. We can help.

REFERENCES

1. Atlassian, *State of Developer Experience Report 2026*. atlassian.com/teams/software-development/state-of-developer-experience-2026
2. Anthropic, *The AI-native SDLC playbook*, August 2026. claude.com/resources/articles/the-ai-native-sdlc-playbook
3. G. Orosz, *Inside OpenAI's agentic software factory*, The Pragmatic Engineer, September 2026. newsletter.pragmaticengineer.com/p/openai-software-factory
4. OpenAI, *Building an AI-native engineering team*. cdn.openai.com/business-guides-and-resources/building-an-ai-native-engineering-team.pdf
5. B. Houck, M. Kanat-Alexander, E. Kalliamvakou, M-A. Storey, N. Forsgren, *CAFE(S): Your Agent Is Only As Good As Its Context*, ACM Queue, 2026. queue.acm.org/doi/10.1145/3847288
6. DX, *State of AI Impact in Engineering: Q3 2026 Report*. getdx.com/report/state-of-ai-impact-in-engineering-q3-report
7. DX, *Measuring developer productivity with the DX Core 4*. getdx.com/research/measuring-developer-productivity-with-the-dx-core-4
8. DX, *Measuring AI code assistants and agents*. getdx.com/research/measuring-ai-code-assistants-and-agents
9. B. Böckeler, *Understanding Spec-Driven Development: Kiro, spec-kit, and Tessl*, martinfowler.com. martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html
10. A. Karpathy, *LLM Knowledge Bases*, April 2026.

Let's Chat

P 1800 983 338

E contactus@shinesolutions.com

www.shinesolutions.com