



WHITE PAPER

AI Development Practice

An informational primer on the shift from traditional, human-authored software development to AI-first development.

REDSHIFTED VENTURES

AI Development Practice · Revised September 2026



AI Development Practice

An informational primer on the shift from traditional, human-authored software development to AI-first development, and an introduction to the Redshifted AI Development Practice.

Executive Overview

- **What AI Development *is*:** AI agents write most of the code, while people set intent, design architecture, and verify. The tools themselves become AI-first (command-line and agent-operable), not human-facing.
- **What AI Development *isn't*:** AI autocomplete added to a traditional workflow gives teams a productivity boost of about 10–15%. Companies that pair AI with end-to-end process change report 25–30%, according to Bain & Company.¹ The difference comes from changing who authors the code and the workflow around it.
- **Why AI Development is a leap:** in our experience, the larger results come when five things change together: leadership mandate, roles, workflow, tooling, and governance. The old structures were built for human-authored code.
- **What AI Development is designed to improve:** security, reliability, repeatability, consistency, quality, speed, and cost, as separate, hand-maintained systems become one connected, self-syncing system.
- **How to adopt AI Development:** prove it on a real “lighthouse” project measured against your own baseline, keeping verification and governance first-class. The Redshifted AI Development Practice delivers this hands-on, on open, self-hostable frameworks. It is the same model Redshifted runs on its own projects.

What is AI Development?

In traditional software development, engineers author code directly: they translate requirements into source code by typing it, line by line. Tools have improved over the decades (better IDEs, autocomplete, code search), but the fundamental model has stayed the same: a human writes the code and all the tools are human-centric and human-facing.

AI Development describes a different model. Here, AI coding agents produce the bulk of the implementation, while humans concentrate on the work that surrounds it: defining intent and requirements, designing architecture, reviewing output, and verifying correctness. The human role shifts from *author* to *director and editor*. Tooling changes to become AI-first, primarily using command-line interfaces rather than human-facing applications.

The difference is concrete: traditional development spreads work across independent, human-facing systems that are maintained by hand and easily drift out of sync, whereas an AI-first setup connects them so agents keep them current.

System	Traditional Development	AI Development
Product definition (PDD)	A hand-written product/design document, updated manually and separately from the code, so it drifts from what is actually built.	Intent and specs live as version-controlled Markdown next to the code; agents build from them and keep them consistent with the implementation.
Ticketing / work tracking	A separate tool where tickets are created, updated, and closed by hand; status lags reality.	Work items live with the code in an agent-readable issue store plus the project board. Agents open, update, and close them as work happens, so status stays current.
Source code	Humans author code line by line in an IDE. Code is the primary artifact, and typing it is the bottleneck.	Agents produce the bulk of the implementation from specs; humans review and validate. Code is generated and verified rather than hand-typed.
Documentation	Written separately after the fact and rarely updated, so it quickly goes stale.	Generated and maintained by agents and linked to the code that backs it; drift is detected and auto-repaired, so docs stay current.
Releases / changelogs	Release steps and changelogs are assembled by hand, which is error-prone and inconsistent.	Automated pipelines with verification gates; release notes and changelogs are generated from the work ledger, so they are consistent and repeatable.
Testing & CI	Tests written and run in a separate phase, often lagging the code.	Automated tests and verification/security gates run on every change as part of the loop; agents write tests and self-review before a human does.
Knowledge & onboarding	Tribal knowledge in people's heads and scattered wikis; onboarding waits on someone's availability.	A self-maintaining knowledge base with an agent on hand to answer questions; new people get productive by self-service.

What *Isn't* AI Development?

Bolting AI tools onto a traditional development structure is not AI Development. It is easy to conflate two very different things:

- **AI as autocomplete (code completion).** Using AI to speed up otherwise traditional programming: suggesting the next line or function inside an unchanged workflow. It is the common entry point, but it is not AI-first development: it yields incremental gains (about 10–

15% in Bain's research²) and leaves the development model (roles, workflow, and economics) unchanged.

- **A tool bolted onto the old system.** New tools layered onto old roles, tickets, and review habits inherit the old constraints. Individual output may rise, but the constraint moves downstream to review. Faros AI's telemetry on 22,000 developers found median time in review up 441.5%.³

Using AI to write code faster is not the same as changing who writes the code. The latter, with AI as primary author, is what the rest of this document means by AI Development.

Implementing AI Development at your company

It's less about buying tools than about changing several parts of how an organization builds at once, on real work. The reason gradual adoption tends to stall is structural: traditional team structures, workflows, role definitions, and economics were all calibrated for human-authored code, so layering AI onto those unchanged foundations constrains what it can do.

The more substantial outcomes tend to appear only when these five elements change together:

1. **Leadership direction.** A clear organizational decision that AI-first is the intended way of building, with the authority to redesign roles and process accordingly. Without it, established habits tend to reassert themselves.
2. **Roles.** Engineers spend less time typing implementation and more time specifying intent, designing architecture, and reviewing and validating output. The most valuable skills become precise specification and fast, sound judgment of quality.
3. **Workflow.** The unit of work becomes a cycle: a clear specification, an AI agent's implementation, human review, and automated verification. Requirement clarity becomes the primary constraint.
4. **Tooling and infrastructure.** Agentic coding tools, connections to the organization's real systems, reusable internal "playbooks" that encode how the team builds, and the testing, CI, and review gates needed to trust faster output.
5. **Trust and governance.** Code review, security review, and quality gates become more important, not less. When implementation is produced several times faster, the organization's ability to *verify* it determines whether speed is an asset or a risk.

The general point is that these elements are interdependent. Changing one in isolation (new tools but old roles, or new roles but no governance) tends to produce limited or unstable results.

AI Development requires a "leap" rather than a gradual transition

This is why a guide who has already been through the transition matters. The Redshifted AI Development Practice is a hands-on engagement that transfers the operating model described here to a client's own team. Its primary activities:

- **Readiness analysis.** A Repository Analysis Framework assesses the codebase, toolchain, team, and delivery baseline and returns prioritized recommendations for building with AI at

speed, starting with the highest-leverage foundations: strict linting gates and automated quality assurance.

- **Quick-win starter assignments.** Early, low-risk wins on non-critical systems, such as a reproducible task driven through a shared control interface (a command runner such as Just) and a bounded API integration, to build confidence and a repeatable pattern.
- **Moving the five levers together.** Leadership mandate, engineering roles, workflow (spec → agent build → human review → automated verification), tooling and infrastructure, and trust and governance, changed together rather than in isolation.
- **A real “lighthouse” build.** The model is tested on a production feature, built by the client’s own engineers and led day-to-day by a client-designated Delivery Lead, with a Redshifted advisory architect pairing in scheduled blocks. It is measured against the client’s own baseline, never a sandbox.
- **Review-layer redesign.** A common failure mode of AI adoption is generation outrunning review,⁴ so the review process is rebuilt for agent-speed work: comprehension gates, pull-request-size discipline, agent-authorship tagging, and automated verification that carries the load line-by-line review once did.
- **The engineer transition, managed.** Leadership and engineer sessions reframe the work so engineers become architects, editors, and validators, and requirement clarity becomes the craft. Because moving from writing code to directing agents is a role change, it is run as a change program, with protected paths for engineers still building trust in the model, and hard guardrails against the “nobody has read this code” failure mode.
- **A playbook and skill library.** The team’s own standards and conventions are codified into reusable playbooks and skills, so the approach is repeatable and owned in-house.
- **Governance by design.** Data handling, IP, security review, and applicable data-protection law are established up front, not retrofitted.
- **Scale and capability transfer.** The model is rolled to a second team, internal champions are trained, and quality and governance gates are hardened, so the client can run it without the vendor.

Delivery is by a small, senior team (an engagement lead, a project manager, and an advisory architect) working one-to-one with the client, primarily remote, on open and self-hostable frameworks. Rather than a full-time embed, the architect pairs and coaches in scheduled, high-leverage blocks while a client-designated Delivery Lead runs the day-to-day, so the capability lands in-house rather than departing with the vendor. This is the same operating model Redshifted runs for its own clients, described in greater detail in the sections below.

Not everything waits for the leap. Alongside the lighthouse, AI-first practices can be introduced into the existing workflow as small, reversible entry points: fundamentals enablement, an automated code-review layer on the current pull-request pipeline, agent-assisted backlog triage, or well-scoped issue-to-PR workflows under existing review gates. On their own these sit in the modest-gains territory typical of tool adoption without workflow change; their value is building confidence, generating a team’s own evidence, and revealing where the bottleneck is. They complement the lighthouse rather than replace it: the on-ramps build confidence, and the lighthouse changes the operating model.

Traditional vs. AI Development, by outcome

Outcome	Traditional Development	AI Development (what we aim for)
Security	Security review is a separate, late-stage phase, done by hand and easily skipped under deadline.	Security scanning and gates run automatically on every change; review becomes first-class precisely because generation is faster.
Reliability	Depends on individuals and manual steps; inconsistent local environments make outcomes hard to reproduce.	Automated verification and CI gate every change; standardized, reproducible environments make behavior predictable.
Repeatability	Work is often one-off; results are hard to reproduce and live as tribal knowledge.	Reusable specifications and playbooks, plus named command recipes, so a result can be demonstrated and re-run rather than argued about.
Consistency	Tools, setups, and conventions drift between people and over time.	Standardized OS and tooling and a single control surface; humans and agents invoke exactly the same commands.
Accessibility	Requires engineers; systems are human-facing GUIs aimed at specialists.	Text- and command-line-first with Markdown legible to everyone; non-engineers can drive the system via chat or the same commands.
Quality	Manual review and a separate QA phase; quality varies with individual diligence.	Automated tests and review gates on every change; agents write tests and self-review before a human does.
Speed & throughput	Output scales roughly linearly with headcount; typing the implementation is the bottleneck.	Agents produce the bulk of the implementation, so a small team can carry more of the roadmap.
Economics	Cost scales with headcount.	Cost shifts toward compute/subscription plus a smaller, more senior team.
Traceability & audit	History is scattered across tools and manual notes.	Named, repeatable recipes and a decision/activity ledger keep the work legible and auditable.
Memory & context	Context lives in people's heads and is lost between sessions.	A durable shared memory persists issues, decisions, and state across sessions, for both agents and people.
Interfaces & tooling	Human-facing, graphical tools that assume a person at a screen.	Programmatic, scriptable, command-line-first tools chosen for how well an agent can operate them; Markdown for input and output.
The engineer's role	Engineers author code line by line.	Engineers act as architects and editors: specifying intent, reviewing, and validating. Leverage comes from framing problems and judging output.

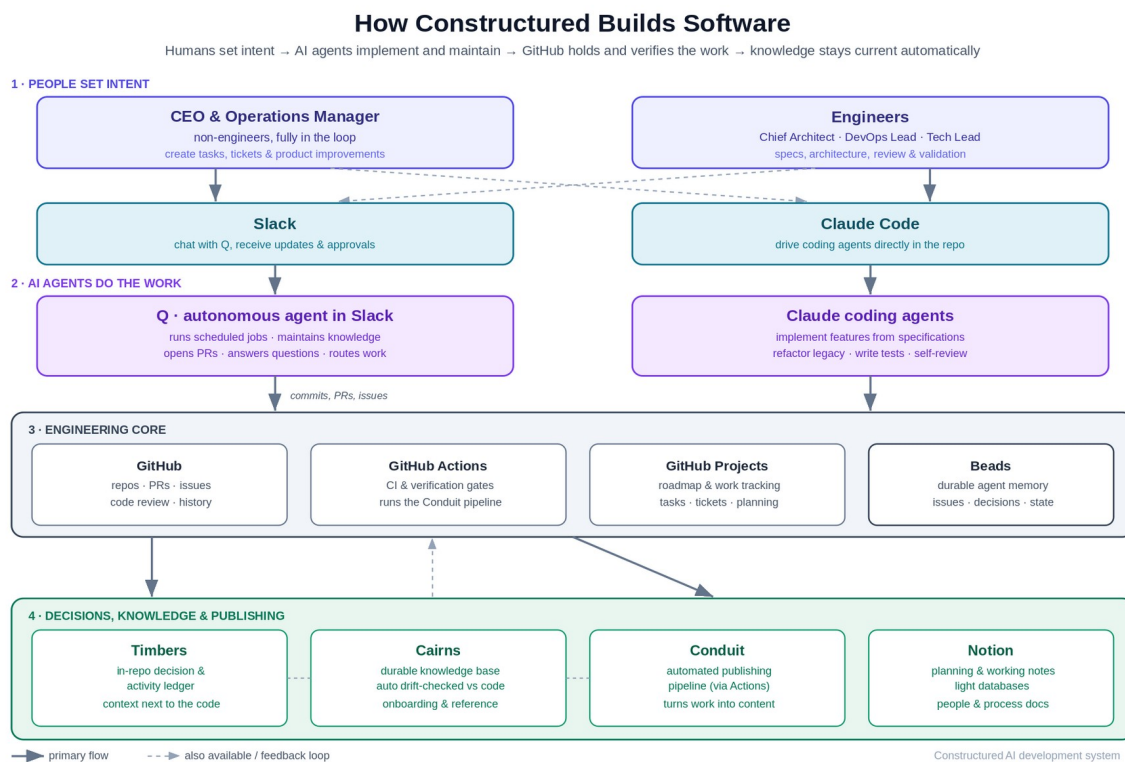
An illustrative example: Constructured

Constructured Inc. (<https://constructured.com>), an early-stage software company, was built to the specifications of the Redshifted AI Development Practice from the outset rather than transitioning to it. A small core team (a chief architect, a DevOps/infrastructure lead, a part-time tech lead, and a part-time product manager) stood up a multi-product roadmap and ran a live pilot with a design partner within months of incorporation.

In day-to-day practice, agentic AI tools are a default part of the workflow rather than an experiment. Engineers specify requirements and architecture; agents produce implementations; engineers review and validate. The notable characteristic is throughput-per-person: a small team, half of it part-time, carrying a multi-product roadmap and a live pilot at the same time. The point is not scale (Constructured is early-stage) but that the model is accessible to any small team that adopts it deliberately.

Inside the system: how the tools fit together

The sections below describe Constructured's setup concretely, as one worked example of an AI-first development system. Each tool has a distinct job, and the work flows in a consistent direction: people express intent, AI agents do the work, GitHub holds and verifies it, and a knowledge layer keeps itself current, with feedback loops where they are useful.



People set the intent. Engineers work primarily through Claude Code, the coding-agent layer in the repository. Non-engineers, including the CEO and the operations manager, work either through Slack or directly in Claude Code. In both cases the people define what is needed: tasks, tickets, and product improvements.

Two on-ramps. *Slack* is where the team talks to Q (our name for a locally hosted OpenClaw implementation) in plain language and receives updates and approvals. *Claude Code* is where coding agents are driven directly against the codebase.

AI agents do the work. *Q* is also surfaced as an autonomous agent that lives in Slack; it runs scheduled jobs, maintains the knowledge base, opens pull requests, answers questions, and routes work. *Claude coding agents* implement features from specifications, refactor existing code, write tests, and self-review before a human does.

GitHub is the engineering core. *GitHub* holds the repositories, pull requests, issues, code review, and history. *GitHub Actions* runs continuous integration and deployment (CI/CD) and automated verification gates, and also runs the Conduit publishing pipeline. *GitHub Projects* tracks the roadmap, tasks, and tickets.

A knowledge layer that maintains itself. *Timbers* is a lightweight decision-and-activity ledger stored alongside the code, keeping engineering context next to the work. *Cairns* is the durable knowledge base used for onboarding and reference; its articles are linked to the code that backs them. When the code drifts, Q detects the gap and opens a repair pull request, so documentation stays current without manual upkeep. *Conduit* is an automated pipeline that turns engineering work into published content. *Notion* holds planning, working notes, light databases, and people-and-process documentation.

The point of the picture is not the individual logos but the shape: a single, connected loop in which the same system that builds the software also documents itself, verifies itself, and can be operated by technical and non-technical people alike.

What AI Development makes easy

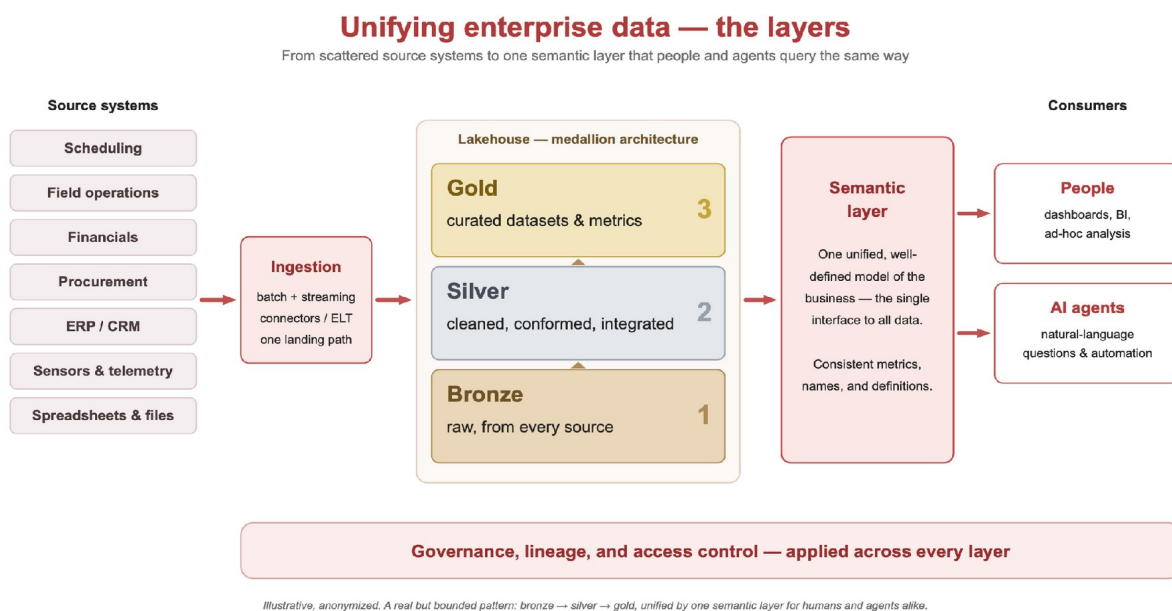
Onboarding is fast, including for non-engineers. A part-time tech lead joined in mid-May 2026 and was soon merging infrastructure changes for the whole team, including a change that moved the development environment's tooling to a single version manager, with migration guidance for the other developers. A product manager, a non-engineer, joined a week later and came up to speed largely by reading the Cairns knowledge base (starting with its guide) and asking Q questions in Slack. Because the knowledge base is current and an agent is on hand to answer questions, a new person does not have to wait on someone else's calendar to become productive.

Changing how everyone works is a pull request, not a project. Because the toolchain is defined in the repository and managed centrally, a single merged change updates every developer's environment at once. The version-manager migration above landed that way, with automated checks keeping everyone consistent afterward. Tweaking the system for all users is routine rather than disruptive.

Anyone can drive the system, through Slack or directly. The CEO and the operations manager use the same system the engineers do. Through Slack they direct Q in plain language; or they work directly in Claude Code. Either way they can create tasks and tickets, file and refine product improvements, reorganize documentation, and move work forward, without writing code themselves. In practice this has ranged from leadership updating a knowledge-base article straight from a Slack conversation, to the operations manager using Q to structure new Notion pages and organize the company’s tool catalog. The system meets people where they are.

A second example: unifying enterprise data

A very different engagement shows the same operating model applied well outside green-field software. An established construction and infrastructure company faced the classic problem of a large, operations-heavy enterprise: its data lived in a dozen disconnected systems (scheduling, field operations, financials, procurement, and more) with no single, trustworthy way to ask a question across all of it. The Redshifted AI Development Practice was applied here not to ship an application but to build a data foundation: a unified semantic layer over the organization’s data. It is a medallion lakehouse that ingests from every source, with a semantic model on top that lets people, and agents, query the whole enterprise through one interface.



The work was done AI-first. A small, senior team specified the domain model, the ingestion contracts, and the semantic definitions; AI agents produced the bulk of the pipeline and transformation code across the bronze, silver, and gold layers; and the humans concentrated on modeling judgment, review, and verification. The work was carried by a handful of people, on the client’s real systems and measured against the client’s own reporting.

What began as a bespoke internal project proved general enough to become a repeatable offering: the same “one place to reach all of an enterprise’s data” pattern is now being productized so that

other operations-heavy organizations can adopt it. The example makes three points. The AI-first model applies to data and analytics work, not only application code. It suits established enterprises with sprawling legacy systems, not only startups. And an internal project, done this way, can produce reusable intellectual property that the organization owns.

The Redshifted Advantage

The market for AI-first consulting tends to fall into two camps: tool vendors that center the engagement on their own proprietary, subscription coding tools, and larger consultancies that lead with a strategy deck, which can leave a client with impressive demos but little enterprise-wide change.

Redshifted's AI Development Practice sits between them: a small, senior team that transfers the outline of the operating model it runs itself, and then steps back. It's about helping you discover what is essential to your enterprise, and then training your teams to work AI-first within the boundaries that you have set.

- **Lived, not just studied.** We run this practice ourselves: Constructured was built this way from day one by a small, senior team. What Redshifted transfers is an operating model in daily use, not a theory from a slide deck.
- **Open frameworks, no lock-in.** The approach is built on free open-source, self-hostable tools (OpenClaw, together with the Cairns, Timbers, and Beads patterns) and not a heavyweight, proprietary, subscription-based methodology or platform that ties a team to one vendor's roadmap and price. The client owns what is built and can run it without the vendor.
- **Tailored, culturally and technically.** The model is calibrated to the client's stack, conventions, and how the team actually works, starting on low-risk repositories and adapting to that reality rather than imposing a one-size-fits-all program. After understanding which variables need to be adjusted, you decide how far to take each change.
- **One-to-one, hands-on delivery.** A small, senior team works directly alongside the client's team: pairing with the principals, not a course, a chatbot, or work handed to junior staff.

The intended result is capability that stays in-house: the client's own playbook, people, and infrastructure, proven against the client's own baseline, with no ongoing dependency on the vendor.

In short: lived, open, tailored, and hands-on. It is an AI-first operating model patterned after Redshifted's best practices, built on open, self-hostable frameworks, calibrated to your stack and culture, and delivered one-to-one by our principals. We have already worked through the hard parts, so your team can adopt AI-first development with that experience behind it.

To find out if the **Redshifted AI Development Practice** can benefit your enterprise, please contact info@redshifted.io to schedule a free initial consultation.

Redshifted Ventures LLC | 11001 W 120th Ave., Suite 471, Broomfield, CO 80021

Appendix A: Common questions and considerations

What results should we expect, and how quickly? Expect a J-curve, not an overnight jump. DORA's 2026 report, *The ROI of AI-Assisted Software Development*, describes a temporary dip while workflows, roles, and review practices adapt, followed by gains once the new operating model takes hold.⁵ Its worked example, a 500-engineer organization, assumes a 15% productivity dip for about three months and arrives at roughly 39% first-year ROI, with payback in about eight months. DORA calls its own figures "a high-uncertainty estimate meant to spark a conversation," and finds that the greatest returns come from the underlying organizational system, not the tools. It's worth knowing the base rate, too: MIT NANDA's 2025 study found that about 95% of enterprise generative-AI pilots showed no measurable P&L impact, though its method (52 interviews and a narrow definition of success) has drawn criticism.⁶ A "lighthouse" engagement, measured against your own baseline, is built to avoid the failure modes behind those numbers and to shorten the dip.

Is AI-written code good and secure enough? Quality and security depend on the verification around the generation. In our practice, fast generation is paired with review gates, automated testing, and security review on every change. DORA's worked example shows why: it assumes the change failure rate rises from 5% to 6% after adoption, and counts that instability as a cost against the return.⁷ We treat speed without verification as the main risk.

How does it apply to legacy systems and technical debt? In our experience, reading and refactoring existing code is one of the stronger applications of AI agents, so legacy systems are often where early benefits appear.

How does it affect engineers and team morale? In our experience, engineers respond well when the change is framed and managed as amplification of their work rather than replacement. Roles shift "upward" toward architecture, specification, and review. We see engineer buy-in as decisive in whether an adoption succeeds.

Does an AI-first team forget how to build? No, when handled deliberately. The value of typing implementation is falling toward zero, but the skills a strong engineer brings can be upgraded rather than lost. The defensible moats are product judgment, domain and industry expertise, experiential taste, precise specification, verification and governance, and systems thinking. All of these compound with experience, and so engineers move from authors to architects and editors. The one caveat is being intentional about how less-experienced engineers build fundamentals, so that their judgment is exercised continuously rather than allowed to erode.

Does it only work for startups? The model is most visible in startups because they can adopt it without legacy process, but the underlying practices apply to established organizations as well. We recommend validating the approach on an internal project and measuring it against the organization's own baseline rather than relying on general claims.

What about intellectual property, data protection, and compliance? These are treated as design questions settled up front: what data the tools may access, where IP resides, and how output is reviewed and attributed. In practice that means a few concrete commitments. Everyone on the engagement is bound by confidentiality (NDA) and invention-assignment (CIIAA) terms, so work product and inventions are assigned to the client, not retained by the vendor. A core part of the

setup is ensuring your code and data are not used as training material by any AI model provider. That means configuring the accounts, plans, and terms accordingly and verifying it. For organizations subject to data-protection laws such as the GDPR, or seeking security attestations like SOC 2, this governance work is positioned at the start of an adoption rather than handled afterward.

Notes

1. Bain & Company, “From Pilots to Payoff: Generative AI in Software Development,” in *Technology Report 2025* (Boston: Bain & Company, September 2025), <https://www.bain.com/insights/from-pilots-to-payoff-generative-ai-in-software-development-technology-report-2025/>; figures as reported in Dan Robinson, “AI Coding Hype Overblown, Bain Shrugs,” *The Register*, September 23, 2025, https://www.theregister.com/2025/09/23/developers_genai_little_productivity_gains/.
2. Bain & Company, “From Pilots to Payoff.”
3. Faros AI, *The AI Engineering Report 2026: The Acceleration Whiplash* (Faros AI, April 2026), <https://www.faros.ai/blog/ai-acceleration-whiplash-takeaways>.
4. Faros AI, *AI Engineering Report 2026*.
5. DORA, *The ROI of AI-Assisted Software Development* (Google Cloud, 2026), <https://dora.dev/ai/roi/report/>; summarized in Matt Saunders, “New DORA Report Claims Strong Engineering Foundations Drive AI Return on Investment,” *InfoQ*, May 11, 2026, <https://www.infoq.com/news/2026/05/dora-roi-ai-assisted-dev-report/>.
6. MIT NANDA, *The GenAI Divide: State of AI in Business 2025* (MIT NANDA, July 2025); for the critique of its method, see R. Scott Raynovich, “Why We Don’t Believe MIT NANDA’s Weird AI Study,” *Futuriom*, August 26, 2025, <https://www.futuriom.com/articles/news/why-we-dont-believe-mit-nandas-werid-ai-study/2025/08>.
7. DORA, *ROI of AI-Assisted Software Development*; Saunders, “New DORA Report.”



GET STARTED

Ready to make the leap?

To find out if the Redshifted AI Development Practice can benefit your enterprise, please contact info@redshifted.io to schedule a free initial consultation.

Redshifted Ventures LLC

11001 W 120th Ave., Suite 471, Broomfield, CO 80021

redshifted.io · (303) 625-9487