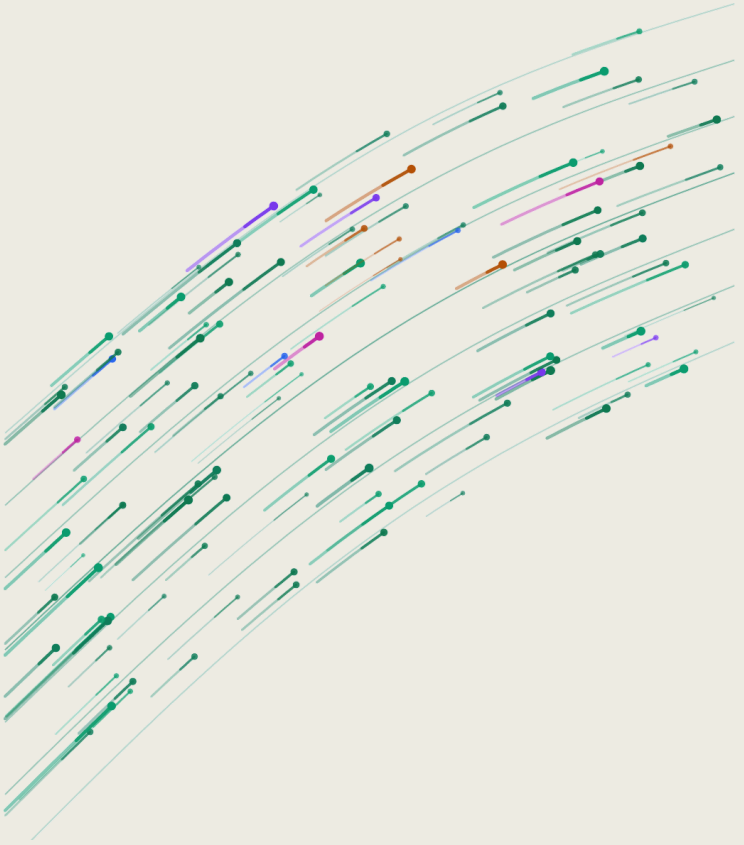
Moderne

The Practical Agentic Coding Playbook

Getting from Mandate to Measurable Results

> The organizations getting agentic engineering right aren't the ones deploying fastest. They're the ones **governing spend and outcomes from day one.**



Individual AI gains aren't compounding into enterprise velocity

Is there shipping stagnation hiding in your AI rollouts?

Every developer survey delivers the same headline. AI makes individual engineers faster. Faros AI's 2026 study of 22,000 developers found **21% more tasks completed** per engineer and **98% more PRs merged**. The 2025 DORA report found 90% of technology professionals now use AI, and 80%+ report higher individual productivity. The lift is real, and it shows up across the work: generating features, writing tests, refactoring services, drafting documentation, building integrations.

At the organization level, the picture is different. Code still moves through a human-paced SDLC built for one repo at a time. Review queues, governance, integration drag. None of it moved. Faros' telemetry across those same 22,000 developers names the result the **Acceleration Whiplash**: bugs per developer up 54% and incidents per PR up 242%.

The DORA 2025 report puts it more directly: AI adoption correlates with increased software delivery instability. Speed at the keyboard is not translating to speed to production.

Boards are asking the obvious question. *Are we seeing the ROI from our investments in AI?* In most orgs, the honest answer is "unevenly." Non-deterministic generation produces uneven code quality. Gaps show up at every handoff: local churn, review backlogs, code that doesn't fit once merged, tests that miss what the agent missed. The pipeline was built for human throughput; AI is feeding it at machine speed. **The paradox isn't that AI doesn't work. It's that the system around it wasn't built for AI.**

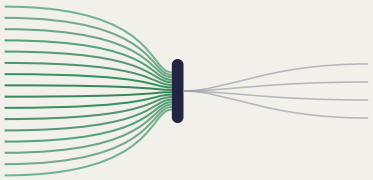
INDIVIDUAL

+55%

Developer productivity

generative AI lift, per engineer

HUMAN-PACED SDLC



review · governance · integration

ENTERPRISE

Bottlenecked

Business velocity

gains absorbed before they ship

Three walls between AI and enterprise ROI

The walls that absorb agent gains: upstream, midstream, downstream.

The reason individual productivity isn't compounding into enterprise velocity is not mysterious. Three walls sit between AI-paced developers and AI-paced delivery. They show up at every enterprise scaling agentic coding: upstream of the agent, inside the agent loop, and downstream of the agent.



WALL 01

Tech debt

Enterprise codebases carry decades of accumulated decisions: dead code paths, inconsistent patterns, framework versions three majors behind, vulnerability remediation deferred. Agents hit this wall hard, generating confident code that doesn't fit, missing cross-cutting dependencies, propagating legacy patterns. And AI doesn't just inherit tech debt, it accelerates it. DORA 2025 reports PR size up 51% per developer; bigger PRs hide more decisions, and more land unreviewed.

WALL 02

Cost & risk

Inside any single agent session, roughly 60% of tokens go to discovery: grep, glob, search, file reads. The work is non-deterministic, so spend swings up to 30x across sessions. The risk story is the same root cause: reason over the wrong context and the agent produces code that's almost right but not quite, the 66% of developers who reported output that compiles, runs, and fails review.

`10+ loops` `2.5% signal-to-noise` `30x variance` `63% wasted tokens`

WALL 03

The SDLC bottleneck

Every change an agent produces eventually meets the SDLC bottleneck found in the outer loop heading to production. Code review, integration testing, security review, change approval are all paced for one human, one repo, one PR at a time. DORA 2025 reports PR review time up 441%. When AI-paced output meets human-paced shipping, every review queue becomes a chokepoint, and quality, accuracy, and cost all suffer.

Scale the walls to scale your agentic output.

Harnessing agentic coding outcomes

Most enterprises are already building harnesses. The differentiator is what they're built on.

Practitioners are already responding to these walls: sub-agents and orchestration patterns, scaffolding from every major model provider, internal harnesses built by enterprise platform teams. Harnessing is a recognized direction. The harder question is what goes inside one.

A harness is only as good as the data it runs on. Most efforts today wrap the same primitives in newer packaging: grep, file reads, text-based search, prompt-driven transformation. The model is still reasoning over noise. The agent is still rebuilding context every loop. That's why so many harnesses produce the same loop counts, the same token bills, and the same almost-but-not-quite-right code as agents alone.

Moderne takes a data-first approach: a substrate underneath everything the harness does, enabling organizations to scale those walls and ship more code.

↑ EVOLVING FAST

Agents are becoming more capable

Powerful, but they need structure, context, and guardrails to act reliably on enterprise code.

↑ MUST KEEP PACE

The SDLC wasn't built for agent velocity

Manual review loops and sequential handoffs become a bottleneck at machine speed.

↓

THE SOLUTION

Deterministic Harness

Deterministic tooling and orchestration working as one system. Agents operate reliably, efficiently across the full SDLC.

DETERMINISTIC AGENTIC HARNESS

Code as structured data, not text.

Symbol-aware, compiler-accurate. The agent reasons over what code means, not what it says. Search returns the right answer in milliseconds, same result every run. Drift is measurable, and AI-generated debt is detectable before it lands.

→ tech debt wall comes down

Context pre-computed, not re-derived.

The read-parse-model loop runs once, offline, with determinism, not on every invocation. Agents skip the 60% of tokens currently spent rediscovering the codebase from scratch.

→ cost & risk wall comes down

Transformation as program, not prompt.

Repeated work runs as auditable programs. Reasoning is reserved for the ~20% that needs it; the other ~80% runs deterministically, same result on ten repos or ten thousand.

→ predictable cost at any scale

Governance built in, not bolted on.

Every agent action, every change, every artifact captured as structured data: queryable, attributable, governable. Work compounds into institutional intelligence, not a review queue.

→ SDLC bottleneck comes down

Three core competencies behind the deterministic harness are what the rest of this playbook is about.

Swap probabilistic for deterministic whenever possible.

Why determinism matters

Probabilistic agents need a deterministic substrate.

Agents are non-deterministic by design. They reason, infer, and generate. It's what enables their autonomy. Agents also work from the least informative representation of software: code as text. Instead of reasoning from an authoritative model of the system, they search files, follow imports, and infer relationships one step at a time. Every task begins by reconstructing context, making understanding probabilistic rather than precise.

That's why you can't build an enterprise operating model on agent inference alone.

Moderne's deterministic harness is built on a precise, reliable, and repeatable understanding of the code itself called the Lossless Semantic Tree (LST). Whether finding every call site, resolving a dependency graph, or applying a complex transformation across an entire portfolio, the harness produces the same result every time — without relying on agent inference. Agents working on top of it reason better because they operate on authoritative code semantics instead of reconstructing meaning from text.

DETERMINISM ORIGINS: YOUR CODE AS DATA

FOUNDATION

Codebase

Raw source files in a repo or on disk. The unstructured starting point for everything.

→

ENRICHED MODEL LAYER

The Lossless Semantic Tree (LST)

Your code as compiler-accurate structured data. Lossless, nothing discarded, down to whitespace and comments. Semantic, every type, symbol, and dependency fully resolved. Tree, structured and traversable programmatically. Parsed once, queried many times, by humans and agents alike.

→

TRANSFORMATION LAYER

Recipes: programs, not prompts

A reusable program that traverses the LST, visits nodes matching a pattern, and either marks them (search) or transforms them (remediation). Same input, same output, every run. Composable, and they emit data tables as a byproduct. Agents author them too.

Together, the LST and recipes give agents what they've been missing: a deterministic substrate to reason about and to use for discovery and transformation. Agents still loop, but the loops are short. Context stays clean, inputs precise. The 60% of tokens that go to discovery in a typical session can be reclaimed for the 20% of work that actually needs reasoning.

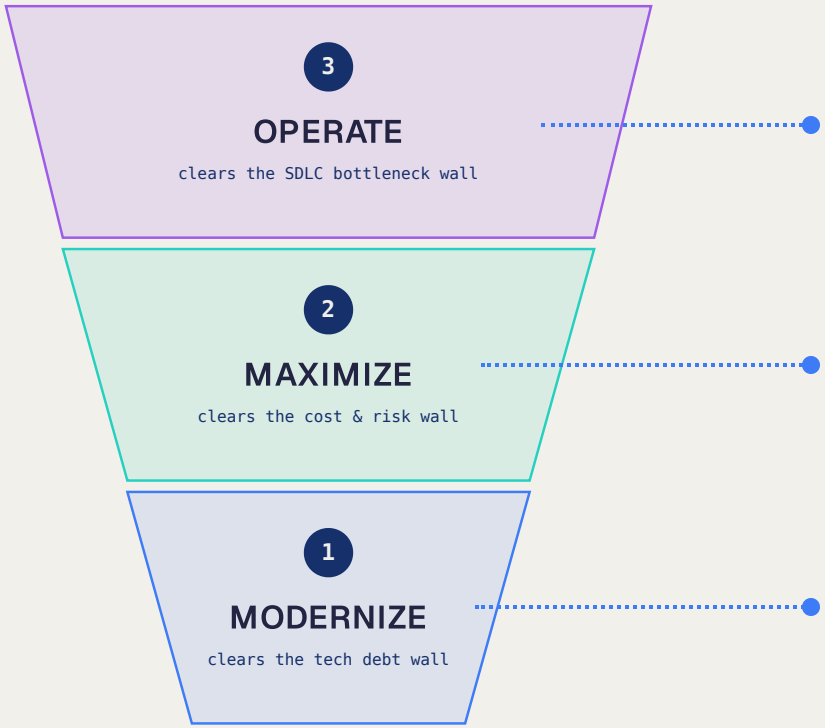
Determinism is the foundation. Everything else is built on top.

Three layers of agentic competency to turn on

Modernize, Maximize, and Operate are competencies you build and keep running. Moderne helps with all three.

Now is where the real work begins. Operationalizing agentic coding isn't a journey with a finish line; it's ongoing competencies that require continuous investment as your codebase, agents, and SDLC evolve. The codebase needs to stay agent-ready. The agents need to stay efficient and accurate. The systems around them need to scale as more of the work moves to AI.

Moderne is the deterministic infrastructure underneath your agentic coding. We ground Claude Code, Copilot, Codex, Cursor, and whatever comes next in accurate search, context, and transformation, scaling change across thousands of repos and billions of lines of code.



LAYER 3

Govern and scale the agentic system

Orchestration across thousands of repos, holding AI-generated change to senior-engineer standards. Intelligence on code changes, agent transcripts, and artifacts becomes the guide, ensuring continuity and smarter agent performance over time.

LAYER 2

Keep agents efficient and accurate

The agent loop runs here. Deterministic discovery, pre-computed context, and programs agents can call keep token spend per outcome trending down and wrong-context-wrong-code from becoming the default.

LAYER 1

Keep the codebase agent-ready

The codebase has to be data agents can reason over, and stay that way. CVEs land weekly, frameworks deprecate, AI-generated debt piles up fast. Modernize keeps the foundation current, secure, and consistent. The tech debt wall rebuilds itself the moment maintenance stops.

Modernize: Build a codebase agents can rely on

AI doesn't fix a fragile codebase. It exposes one.

Modernize is the standing competency of keeping enterprise code in a shape agents can reason over. The work itself isn't new. It's the battle engineering orgs have been waging for decades: framework upgrades, dependency updates, dead code removal, deprecated API migrations, vulnerability remediation. Every engineering org has a list.

What's different now is the pace. Humans could defer maintenance because the rate of change was slower. In the AI era, deferred work is catching up, and agents add to the pile every day. They inherit the assumptions of the code around them and propagate weaknesses forward. Security is where the gap is most visible, and AI just widened it. Models like Anthropic's Mythos are surfacing vulnerabilities at exponential rates, yet enterprises already can't keep up, with one study citing **45% of vulnerabilities unpatched** a year after disclosure.

Modernize is both a precondition and a discipline. Without it, agents spend their tokens reasoning over noise, generate code that churns, carry over vulnerabilities, and quietly amplify the patterns you were trying to retire. As a discipline, it's the regular practice of quality sweeps, dependency reviews, and coordinated transformation campaigns that keep AI-generated debt from compounding.

Measuring success comes down to scope and reach. Scope is how complete each migration is: version bump, dependency updates, API changes, deprecations, all of it. Reach is how much of the estate is actually covered.

The "Moderne way" — horizontal change **80%**

100 of 125 tasks modernized business-wide



The traditional way — one app, one repo **3.2%**

4 of 125 tasks modernized business-wide



WHERE ARE YOU ON MODERNIZE?

How many versions of a main framework run across your estate?

- ○ **One** — uniform estate. Agents have consistent ground to work from.
- ○ **Two to five** — some drift. Agents hit inconsistencies that affect reasoning.
- ○ **More than five** — substantial fragmentation. Agents struggle to apply patterns consistently.

Keep AI-generated debt from compounding.

WHAT GOOD LOOKS LIKE

Horizontal change is the default.
A framework upgrade or CVE remediation runs as one tracked campaign across the estate.

Similar services actually are similar.
Drift between equivalent codepaths drops toward zero, so agents carry context service to service.

Remediation in hours, not weeks.
A CVE lands, the campaign launches, the fix ships before the next standup.

AI-debt caught at the door.
Surface agent duplication, complexity, anti-patterns immediately, not a year later.

HOW MODERNE HELPS

The Lossless Semantic Tree.
Your code as compiler-accurate structured data: the substrate for search, context, and validation.

10,000+ deterministic recipes.
Auditable programs for migrations, remediation, and cleanup across Java, JS, C#, Python, Go, Scala.

Horizontal change at estate scale.
Recipes run simultaneously across 10,000+ repositories as one coordinated campaign.

Moderne Managed Service option.
Forward-deployed engineers plan and run your campaigns, with SLA-backed outcomes.

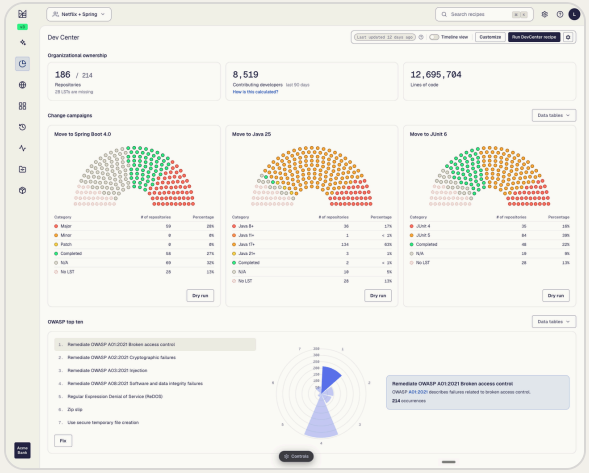
WHAT TO MEASURE

% of estate on supported versions.
The best indicator of foundation health. Trending up monthly means it's working.

% of vulnerabilities fixed estate-wide.
Know how quickly and completely an organization is closing the risk gap.

Code complexity.
Quality metrics for method complexity, class complexity, dependency instability, and test gap risk.

Recipe coverage.
How many ran, landed in PRs, and merged, across how many repositories.



Moderne DevCenter for tracking migration and vulnerability remediation progress across the estate.

A clean codebase is the cheapest token-efficiency optimization you'll ever make.

Maximize: Get the most out of every agent you run

Build productive, less expensive agent loops.

Maximize is the discipline of helping every agent in your org do its best work: fewer wasted tokens, fewer wasted loops, output the team can actually ship. The agent loop runs here. Bad inputs compound into worse outputs, and token waste multiplies with every loop.

The root of the inefficiency is a context gap. Agents infer context from the text they read, file by file, token by token, rather than reasoning over actual, resolved code data. Inference is plausible-sounding but unreliable. Roughly **60% of tokens in a session** go to this discovery work, the most non-deterministic part of the loop, before the bottleneck moves downstream to reviewers validating the output.

The agents you've already deployed, Copilot, Claude, Cursor, and so on, get substantially more efficient and reliable when the substrate underneath them is accurate, resolved data. **Same agents, better data, different math:** loops drop from 10+ to 1 or 2, token spend per outcome trends down, and output is grounded in the codebase rather than guessed.

PURE LLM · JAVA 25 MIGRATION

61,000,000+

tokens

RUNTIME	45 min
COMPLETE	25%

AGENT + MODERNE RECIPES

30,000

tokens

RUNTIME	3 min
COMPLETE	100%

2,000× fewer tokens for the same migration, fully complete.

Maximize is how agents start earning their keep.

THREE MOVES THAT UNLOCK THE LOOP

Move discovery out of the model.
Indexed, symbol-aware search returns the right answer in milliseconds. The 30× variance collapses; every loop starts with a clean signal.

Move context out of the model.
Pre-computed structural context from the LST skips the read-parse-model loop. The agent reasons over what code means, not what it says.

Move repetitive transformation out of the model.
Reasoning is reserved for the ~20% that needs it; the other ~80% runs deterministically.

WHERE ARE YOU ON MAXIMIZE?

Do you know your average token spend per shipped change?

- ○ **Yes** — tracked and trending down. The loop is being managed.
- ○ **Yes, but trending up.** Something is wrong with the inputs.
- ○ **No** — we measure usage, not outcomes. That's the gap.

WHAT GOOD LOOKS LIKE

Agents reach for deterministic tools.
The "stuck agent at 47 iterations" stops happening because discovery has been replaced with deterministic input. Loops drop to 1–2; automated code review and approval is possible.

Token spend per task trends down.
As deterministic tools carry more of the loop, cost per outcome falls and becomes more predictable.

Output quality goes up alongside efficiency.
The agent stops inferring the codebase and starts working from it. Wrong-context-wrong-code stops being the default.

HOW MODERNE HELPS

Trigrep indexed search.
Exact code search across the estate in milliseconds, not grep guesses.

Prethink pre-computed context.
The structural context an agent needs, assembled before it asks.

Recipes agents can call.
Deterministic programs for the repeatable ~80%, reasoning reserved for the ~20%.

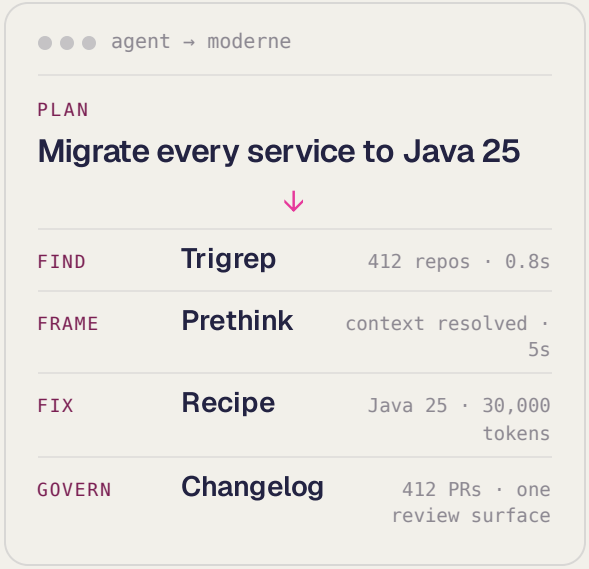
MCP server and CLI.
Your existing agents call all of it through standard interfaces.

HOW TO MEASURE

Tokens per completed task.
The unit that matters is the task: a CVE remediated, a framework upgraded, a test suite generated. Every search, context window, retry, and edit contributes to the true cost of shipping.

Agent loop iterations to resolution.
A clean run resolves in 1–2 loops; a struggling one spirals to 10 or more. Expose where the discovery-to-plan-to-act chain is breaking down. Also a leading indicator for token waste.

Review iterations per AI-generated PR.
The average number of review cycles before merge. Rising review iterations indicate AI-generated code is creating additional downstream work.



Agent tools supporting the agentic loop — plan, find, frame, fix, and govern, each backed by a deterministic tool.

Same agents. Better tools. Different math.

Operate: Govern systems of agents at scale

At enterprise scale, systems of agents need to see the world they're working in.

Operate is the discipline of building the systems agents work in. Artifacts. Change history. Session transcripts. Shared context across teams. Monitoring. It's the world around the agent made legible, so agents can work autonomously with real awareness of what came before and what's running now. A single agent can do impressive things in one repo. The leap to enterprise scale requires an environment agents can read, write to, and act through together.

The questions then become: *"can we trust what the agent did, prove what it did, and govern what it's allowed to do?"* None of those questions can be answered without structured data, not just about the code, but about how it changes, who changes it, and under what conditions. Operate is what makes them answerable and it will become default for engineering orgs, the same way CI/CD did: invisible when it works, painful when it doesn't exist.

Operate measures the system, specifically whether the system is getting better over time. Durability is the right word: do campaigns compound? Are agent actions traceable months later? Is governance keeping pace with throughput, or is it the new bottleneck?

WHAT OPERATE MAKES LEGIBLE

What's been changed.
Multi-repo, multi-SCM PR and commit tracking across organizations. The provenance of every line of AI-generated code, traceable in seconds. Auditors know what shipped; engineering leaders know what their AI investment actually produced.

What's available to build.
The full upstream dependency catalog: every package, every version, all indexed and searchable up front by agents before they make a change.

What agents have done.
Full audit trails for governance, learning, and identifying where the toolset is falling short. Every agent action accountable to a record.

What other agents and engineers know.
Design, UI, engineering working from the same substrate. Agent-to-agent and agent-to-human collaboration built on common ground.

What's running.
The frontier of Operate: connecting signals from production back to the codebase and the agents working on it.

WHERE ARE YOU ON OPERATE?

Can you answer "what did the agents do" in one query?

- ○ **Yes, in one query** — full audit trail, queryable.
- ○ **Only with a forensic dig** — the data exists but isn't unified.
- ○ **Not really** — agent work is opaque once it ships.

Agents that work autonomously, together, at scale.

WHAT GOOD LOOKS LIKE

Visibility is built in.
Engineering leadership can answer "what did our agents do this quarter" in one query, not a forensic audit.

Agentic systems run autonomously.
Outcome-based, not session-based. Deterministic guardrails keep things on track. Efficiency improves with every run.

Institutional intelligence compounds.
Planning improves, execution becomes more predictable, and continuity replaces reinvention across agentic engineering.

HOW MODERNE HELPS

Changelog, Transcripts, Artifacts — systems of record.
Org-aware PR and commit tracking. Full agent session audit trails. Dependency and artifact visibility across the estate. The institutional intelligence layer everything else in Operate is built on.

Moderne Factory.
Work kicks off with a goal (e.g., migrate to Spring Boot 4); specialized agents with deterministic tools coordinate the work; humans steer at decision points. Factory gets smarter and more efficient each run.

Moderne Skills.
Recipe authoring by agents lets the org's specialized knowledge become reusable programs for all to use. Deterministic change propagated by agents.

HOW TO MEASURE

Campaign velocity.
Mean time from goal-set to shipped outcome, measured at the campaign level.

Cost per outcome, trending down.
Total token spend per completed campaign across the estate, not per request.

Share of changes needing human review.
Risk assessment routes each change to the right level of scrutiny, so more determinism means fewer changes land on a senior reviewer's desk.

ARTIFACTS



Dependency Intelligence

impact zones, mapped

TRANSCRIPTS



Agent Session Capture

searchable audit trail

CHANGELOG



Multi-Repo Tracking

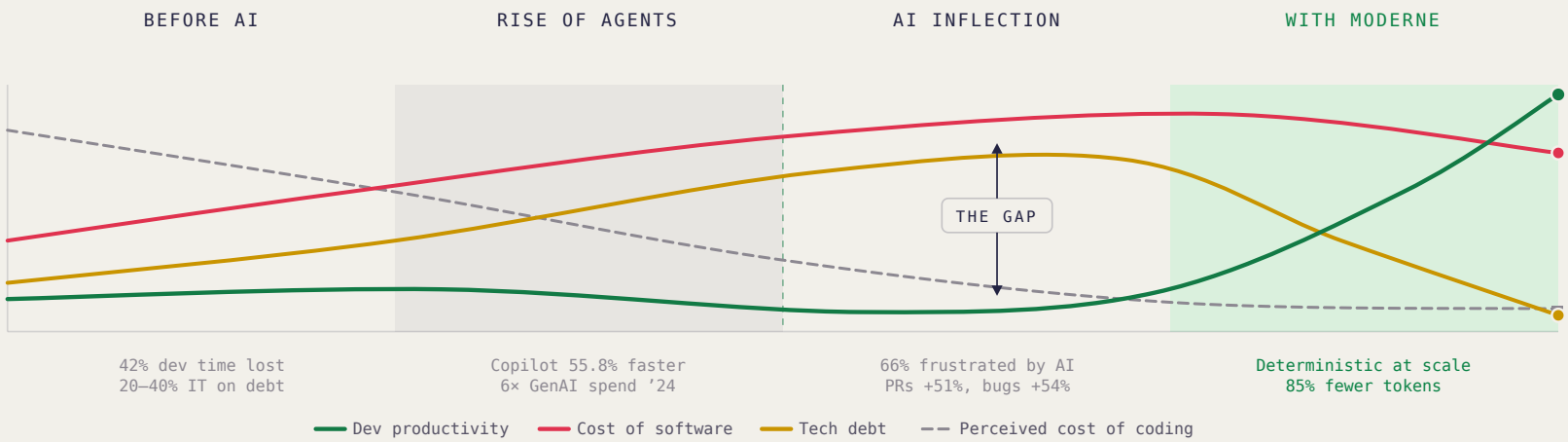
portfolio-wide, all SCMs

Systems of record — institutional memory, the durable asset

Every campaign leaves the next one smarter.

The ROI of getting agentic coding right

AI and determinism deliver compounding returns at scale.



The return on agentic coding isn't a single line item. It shows up as productivity reclaimed, debt paid down, and risk brought under control, and each compounds as the deterministic harness underneath your agents does more of the work. It's time to bend the curve.

Productive delivery, not just productive coding.
Measure shipped outcomes — completed campaigns, review speed, merged PRs.

Tech debt held in check.
Track the share of the estate on current supported versions, month over month. Up means it's working.

AppSec renaissance.
Measure MTTR across the estate. Agent finds it, writes the recipe, runs the recipe, everywhere.

Cost per outcome trending down.
Token spend per shipped change, not per request.

CONTINUOUS MODERNIZATION ROI

Modernization is too often treated as a capital project: six-month engagements, consultants on retainer, then another overhaul when the next framework reaches end of life. Continuous modernization changes that equation. A Spring Boot 3→4 migration that might cost **$500K–$2M** as a consulting engagement finishes in **weeks**. A CVE that would have filled a backlog for months is remediated with a single recipe execution. Continuous dependency upgrades eliminate future big-bang migrations. Modernization becomes a predictable operating expense, with engineers and agents focused on products instead of plumbing.

You'll feel it when shipping stops being the bottleneck.

NEXT STEPS

Start where the wall is highest

Targeted steps toward agentic coding at scale.

Operationalizing agentic coding is a broad initiative that can seem amorphous. These are clear starting points for breaking down the walls in your way. All three areas need attention over time, but the wall costing your org the most right now is where the first move pays back fastest.

Which wall needs your attention first?

Tech debt

Pick one cross-cutting transformation the estate has been postponing: a Spring Boot major upgrade, a Log4j class of vulnerabilities, a deprecated internal framework. Run it as a single horizontal campaign on the Moderne Platform, not a thousand distributed tickets. Measure how complete the change is in each repo, and how much of the estate it reached. The first horizontal campaign tells you more about your modernization posture than any dashboard you have today.

Cost & risk

Give your agents Moderne's deterministic tools through the CLI and MCP server. The agents you've already deployed can call indexed code search, pre-computed context, and proven recipes instead of inferring everything from text. Measure tokens per completed task, loop iterations, and review speed on AI PRs, before and after. It's the proof you'll need for the next budget conversation, and the start of keeping agents their most efficient as the model landscape continues to shift.

The SDLC bottleneck

Start by making change visible. Stand up Moderne Changelog across your active repositories, with multi-repo, multi-SCM PR and commit tracking overlaid on org structure. By the end of the first month, you should be able to answer "what did the agents do, and what did it produce" in one query, not a forensic audit. That single capability is the foundation everything else in Operate gets built on.

Moderne can help put agentic coding to work for your organization, from getting your codebase ready for AI to building agent factories that work autonomously. **Sign up for a working session with our team to start knocking down walls and accelerate your agentic AI adoption.**

moderne.ai/book-a-demo