techahead

Transform Your **Vibe-Coded Application** into an **Enterprise-Ready Solution**

From rapid AI-assisted prototypes to secure, governed, scalable, and supportable software.

1 **Prototype** → 2 **Stabilize** → 3 **Secure** → 4 **Scale** → 5 **Governed service**
The destination



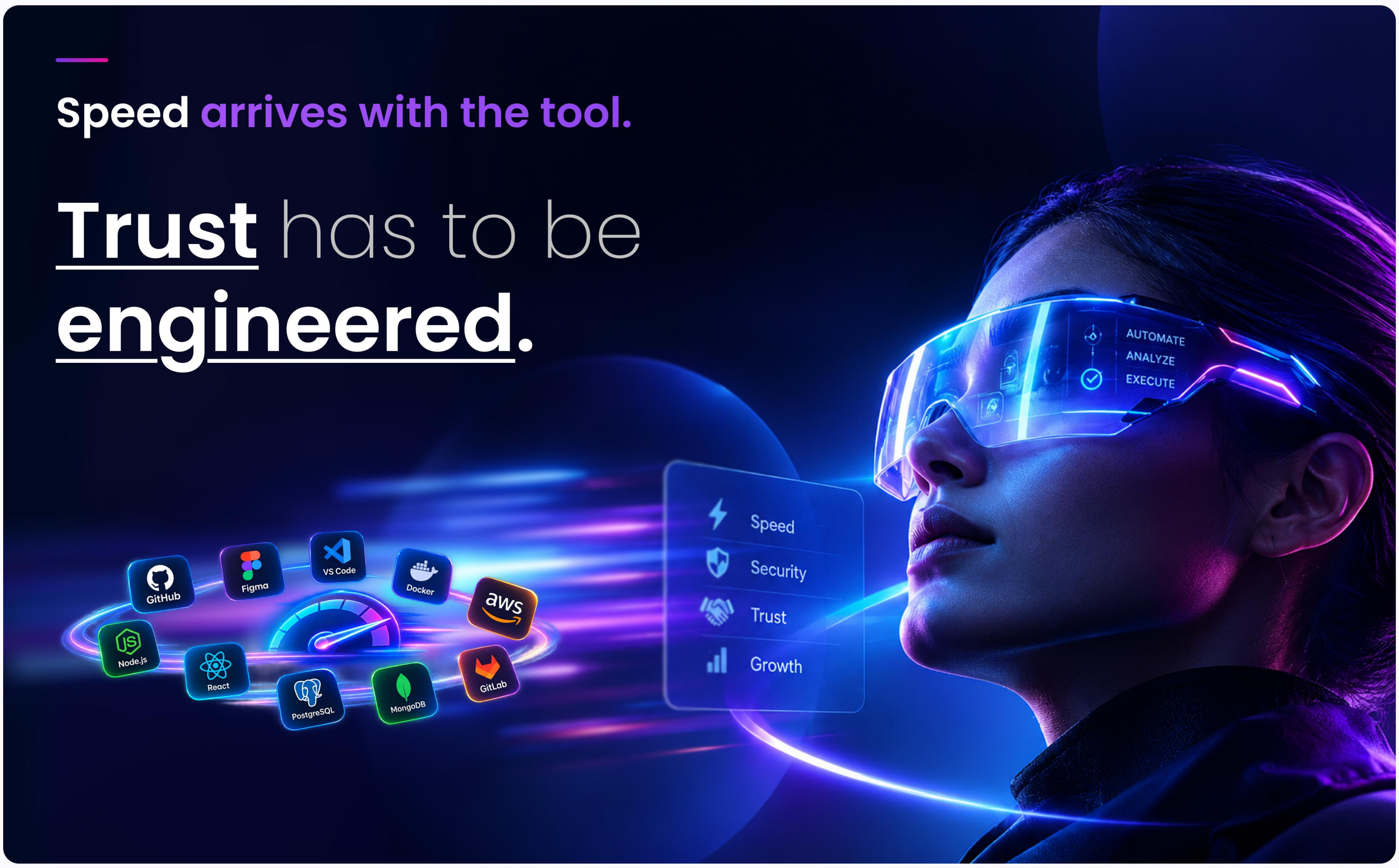
A TECHAHEAD WHITEPAPER · VERSION 1.0 · SEPTEMBER 2026

About this paper

Prepared by TechAhead's enterprise engineering and AI practice, drawing on patterns seen across mobile, cloud, and AI product delivery for startups and large enterprises.

DISCLAIMER

The frameworks, examples, and case studies in this paper are educational. Case studies are anonymized composites and are labeled as illustrative. Security, privacy, and regulatory decisions require review against your organization's own risk profile, contracts, and applicable law.



NAVIGATE THE PAPER

Contents

— Executive Summary 05

01 The Rise of Vibe Coding 08

02 Why Vibe-Coded Software Breaks in Enterprise Environments 12

03 Enterprise Readiness: A Practical Definition 16

04 The Transformation Framework 19

05 Reference Architecture for Enterprise-Ready AI Applications 24

06 Security, Privacy, and Responsible AI Controls 29

07 Case Studies: Lessons from the Field 34

08 Operating Model: People, Process, and Governance 41

09 Metrics That Prove Enterprise Readiness 45

APPENDICES & REFERENCE MATERIAL

10 Enterprise-Readiness Assessment Checklist 48

11 Recommended 90-Day Transformation Path 51

— Conclusion · Appendices A–E · References · About TechAhead 54

REFERENCE INDEX

Figures, tables, and case studies

Figures

01 Figure 1: The Enterprise Readiness Gap

02 Figure 2: The "It Works" Iceberg

03 Figure 3: The Eight Pillars of Enterprise Readiness

04 Figure 4: Vibe Coding to Enterprise Service Maturity Model

05 Figure 5: The Six-Phase Transformation Framework

06 Figure 6: Enterprise-Ready Reference Architecture

07 Figure 7: Guarded Request Path for an AI Feature

08 Figure 8: Two-Speed Delivery and Governance Gates

09 Figure 9: The 90-Day Transformation Roadmap

Tables

01 Table 1: Prototype Signals vs Enterprise-Readiness Signals

02 Table 2: Transformation Phases, Owners, and Deliverables

03 Table 3: AI Risk to Control Mapping

04 Table 4: Roles and Accountabilities (RACI)

05 Table 5: Enterprise-Readiness Metrics and Starting Targets

06 Table 6: Enterprise-Readiness Scorecard

07 Table 7: Sample Risk Register

CASE STUDY CALLOUTS

Internal Knowledge Assistant · Customer-Facing AI Workflow Agent · Legacy Workflow Modernization

EXECUTIVE SUMMARY

A working demo is evidence that an idea is possible.

It is not evidence that the software is ready for real users, real data, and real money.



Closing that distance is an engineering maturity journey, and in most cases it does not require a rewrite.

AI coding assistants now let a product manager, analyst, founder, or engineer turn a plain-language idea into a functioning application in days. We call the result a vibe-coded application: software built mainly through conversational prompts, where the author steers by outcome rather than by reviewing every line that gets generated.

Organizations are embracing this style of work for sound reasons. It shortens discovery, lets domain experts build their own tools, and gives engineers a faster first draft.

90%

of respondents use AI at work

30%

report little or no trust in AI-generated code

2025 DORA State of AI-assisted Software Development report · Google Cloud summary

> **AI acts as an amplifier: it magnifies the strengths of organizations with sound engineering systems and the dysfunctions of those without them. Speed arrives with the tool. Trust has to be engineered.**

The prototype-to-production gap

We call the distance between "it works on my machine" and "we can run this for the business" the prototype-to-production gap. This paper measures it across eight enterprise-readiness dimensions:

Business alignment:
a named owner, a clear value hypothesis, and a decision on whether the tool deserves investment.

Architecture:
modular, documented design that another team can understand and change.

Security:
identity, least privilege, secrets management, and a secure development lifecycle.

Data and AI governance:
known data flows, managed prompts and models, and evaluation evidence.

Quality engineering:
automated tests and release gates that make change safe.

Reliability and performance:
defined service levels, capacity plans, and recovery procedures.

Operability and support:
monitoring, runbooks, incident response, and a support path.

Compliance and auditability:
evidence that controls exist and work.

> The prototype usually wins on speed alone. Every other dimension is where enterprise investment goes.

The transformation framework

The core of this paper is a six-phase framework: Discover and Triage, Stabilize the Foundation, Secure the Application and its AI Components, Engineer for Quality and Change, Operationalize and Scale, and Govern and Continuously Improve.

A four-stage maturity model (Experiment, Pilot, Production Candidate, Enterprise Service) tells teams how much control each application needs at each point, so small experiments stay light and critical services get full rigor.

The framework draws on established standards rather than inventing new ones: the NIST AI Risk Management Framework, the NIST Secure Software Development Framework (SP 800-218), the OWASP Top 10 for LLM Applications, Google's Site Reliability Engineering practices, and DORA's delivery research.

Expected outcomes

Organizations that apply the framework should expect:

Reduced operational risk, because every production application has an owner, a runbook, and a recovery plan.

A stronger security posture, because secrets, permissions, and AI-specific threats are handled by design.

Faster and safer releases, because automated tests and pipelines replace manual checks.

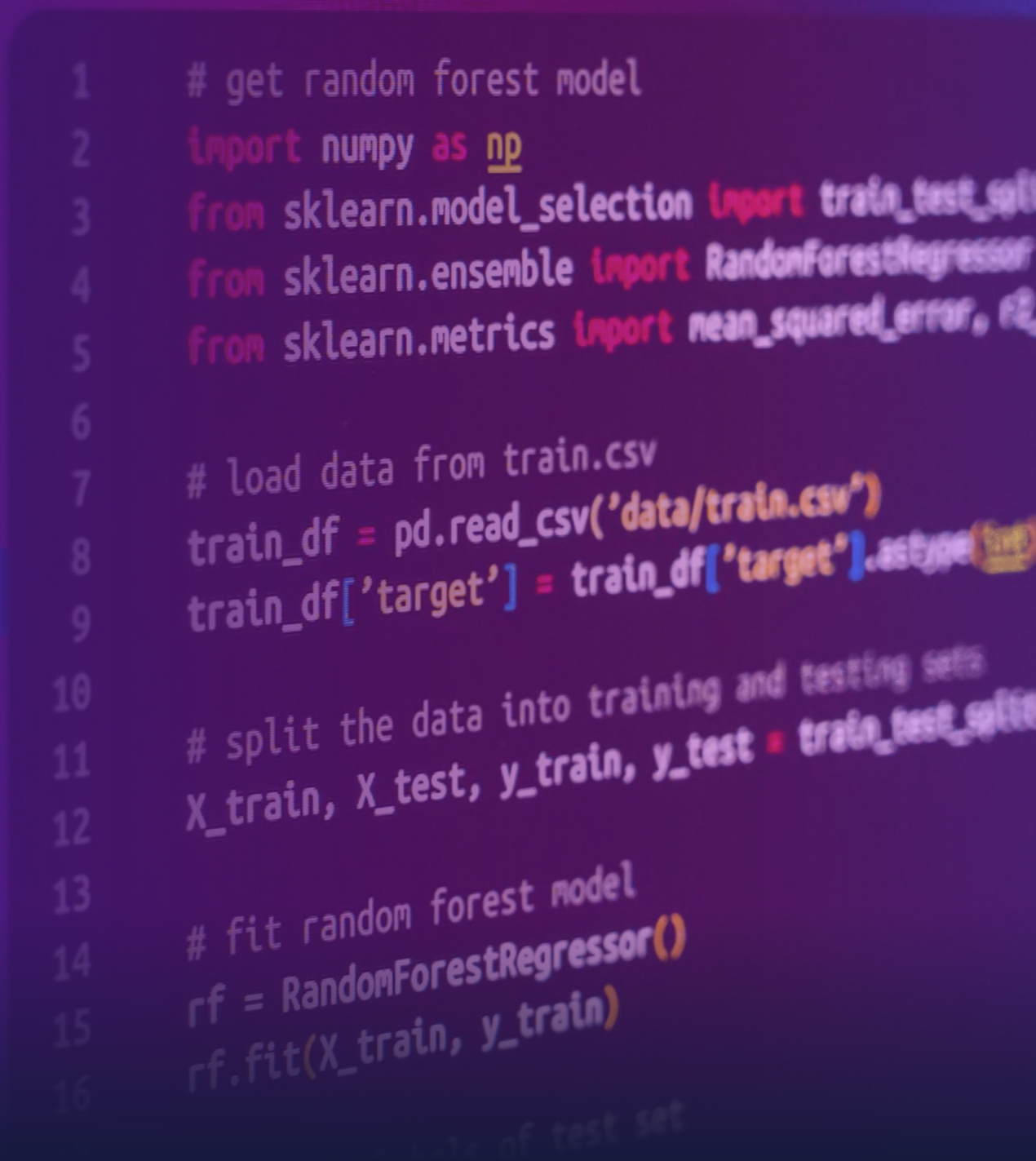
More predictable cost and performance, because model usage, cloud spend, and latency are measured and capped.

Clear ownership and governance, because decisions are recorded and reviewed at defined gates.

01

The Rise of Vibe Coding

Vibe coding has moved software creation from a specialist activity toward a conversational one. That shift is the opportunity, and it is also the source of every risk this paper addresses.



1.1

What is vibe coding?

Describing and directing software through natural language rather than writing code directly.

1.2

Why it is gaining momentum

Falling barriers, rising AI capability, and growing demand from non-technical builders.

1.3

The core tension

Speed and accessibility in direct conflict with security, reliability, and governance.

1.4

The enterprise question

How organisations harness velocity without sacrificing the controls that protect them.

1.1 What is vibe coding?

"Give in to the vibes"

The term was popularized by AI researcher Andrej Karpathy in early 2025 to describe a style of building where you describe what you want, accept what the AI produces, run it, and iterate on the result instead of the code.

01 Conversational prompts drive the work.

The primary interface is natural language, not an editor.

02 AI generates most of the code.

The human reviews behavior far more than implementation.

03 Iteration is rapid.

A feature can go from idea to running screen in minutes.

04 Outcomes come first.

Success is judged by whether it works now, not whether it will be maintainable later.

Separate vibe coding from its neighbors

Low-code and no-code	Platforms lower the barrier to building, but constrain what you can build inside a vendor's governed runtime. Vibe coding produces unconstrained source code that can do anything, including the wrong thing.
AI pair programming	Keeps a trained engineer in the loop, reading and accepting suggestions line by line. Vibe coding often removes that line-by-line review.
Formal software engineering	Starts with requirements, design, and review, then writes code. Vibe coding frequently inverts that order: code first, understanding later.

None of these is inherently better. They carry different risk profiles, and enterprises need to know which one produced the software they are about to depend on.

1.2 Why it is gaining momentum

The momentum is not hype alone.

Vibe coding delivers real value in five ways:

01

Faster prototyping

Teams can test an idea with a working application instead of a slide deck, which makes stakeholder feedback concrete.

02

A lower barrier to building

People who understand a business problem deeply, but never learned a framework, can now produce software.

03

Domain experts build their own tools

A compliance analyst can build the triage tool they have wanted for years without waiting in the engineering backlog.

04

More experimentation

When building is cheap, product discovery can explore more options and kill weak ideas sooner.

05

Developer productivity

Experienced engineers use the same tools to scaffold, refactor, and write tests faster.

1.3 The core tension

Every benefit comes with a matching tension.

Speed of generation	vs	**Confidence in what was generated**	Code arrives faster than anyone can understand it.
Local functionality	vs	**Production behavior**	Software that works for one user on a laptop may fail under concurrency, real data volumes, and hostile input.
Individual experimentation	vs	**Organizational accountability**	A personal tool becomes a team dependency without anyone deciding it should.
AI-generated code	vs	**Secure, maintainable, traceable software**	Nobody can explain why a line exists, which dependency it pulled in, or what data it touches.

DORA's 2025 research reports that AI adoption is now associated with higher delivery throughput, but also with continued increases in delivery instability. Teams have sped up faster than the systems that keep changes safe.

> **A functional demo is evidence of possibility, not evidence of production readiness.**

1.4 The enterprise question

How can organizations retain the speed of AI-assisted development while adding the controls needed for scale, safety, reliability, and compliance?

02

Why Vibe-Coded Software Breaks in Enterprise Environments

Vibe-coded software rarely fails because the AI wrote bad code. It fails because nobody made the decisions that production software requires: who can use it, what data it touches, what happens when it breaks, and who fixes it.

2.1 The prototype-to-production gap

2.2 Common failure modes

2.3 The hidden cost of "it works"

2.1 The prototype-to-production gap

A prototype runs in a forgiving world. It has one user, a local or sample database, a developer's own API keys, generous time limits, and a friendly audience. An enterprise environment removes every one of those comforts at once.

Consider a typical path. A sales-operations lead builds a pricing-approval tool over a weekend. It connects to the CRM with her personal token, stores approvals in a cloud database with default settings, and calls a large language model to summarize deal risk. It works so well that twenty colleagues start using it within a month.

Now the questions arrive. What happens when she changes roles and her token expires? Who can see the approvals of other regions? What does the model provider retain from the deal data it receives? What happens at quarter-end, when usage triples? Nobody designed answers, because nobody knew the tool would matter.

2.2 Common failure modes

Across AI-assisted builds, the same failure modes appear repeatedly:

Unclear requirements and hidden assumptions.
The prompt history holds the only record of what the software is supposed to do, and it contradicts itself.

Fragile dependencies.
Generated code pulls in libraries, sometimes outdated or unmaintained, that nobody chose deliberately.

Hard-coded credentials and API keys.
Secrets sit in source files, environment files committed to repositories, or front-end bundles.

Uncontrolled third-party APIs and models.
Data flows to external services without a contract, a data-processing agreement, or a fallback.

Missing identity and access controls.
A shared login or no login at all, and no distinction between viewing and approving.

Inconsistent data handling.
Personal or confidential data is copied into logs, prompts, and test fixtures.

Weak error handling and recovery.
Failures surface as blank screens or silent data loss rather than clear messages and retries.

No automated tests.
Every change risks breaking something that previously worked, so teams stop changing things.

No logging, monitoring, or incident process.
Problems are discovered by users, not by alerts.

Poor documentation.
Design decisions live in chat transcripts that nobody will read.

Cost spikes.
Unmanaged model calls, runaway agent loops, or oversized cloud resources create surprise bills.

2.3 The hidden cost of "it works"

The visible success of a vibe-coded prototype conceals a larger mass of deferred cost. Figure 2 shows where that cost sits[b].

A working demo proves speed; enterprise readiness covers the rest

	Vibe-coded prototype	Enterprise-ready solution
Speed	Days to a working demo	Fast, with gated releases
Maintainability	Code nobody fully owns	Reviewed, modular, owned
Observability	Console logs, if any	Logs, metrics, traces, alerts
Access control	Shared login or API key	SSO, MFA, least privilege
Test coverage	Manual clicks before a demo	Automated tests in CI
Documentation	Prompt history	ADRs, runbooks, data flows
Compliance	Unknown data exposure	Mapped data, audit evidence
Reliability	Works on the happy path	SLOs, rollback, recovery

Figure 2: The Enterprise Readiness Gap · eight dimensions, before and after

Each cost compounds the others. Rework grows with every feature added to an unstable base. Security exposure turns into compliance risk the moment regulated data enters the system. Enterprise buyers increasingly send security questionnaires before signing, so a product that cannot answer them loses deals it has already won on features.

Table 1: Prototype signals vs enterprise-readiness signals

Teams can use the signals below to tell, quickly and honestly, which side of the gap an application sits on.

Area	Prototype signal	Enterprise-ready signal
Ownership	The person who built it	Named business, technical, data, and security owners
Authentication	Simple login or shared account	SSO, MFA, role-based access, user lifecycle management
Secrets	Keys in code or environment files	Central vault, rotation, scoped credentials
Security	Manual checks	Secure SDLC, vulnerability scanning, threat modeling
Reliability	"Works during the demo"	Defined SLOs, monitoring, alerting, recovery procedures
Testing	Ad hoc manual testing	Automated unit, integration, security, and regression tests
Documentation	Prompt history or informal notes	Architecture, runbooks, APIs, data flows, decision records
AI controls	The prompt worked once	Evaluations, guardrails, versioning, human escalation
Cost	Paid on a personal or team card	Budgets, quotas, per-transaction cost tracking

> An application with mostly left-column signals is not a failure. It is a prototype, and it should be treated as one until someone decides it deserves more.

03

Enterprise Readiness: A Practical Definition

An application is enterprise-ready when the organization can depend on it, change it safely, and prove how it behaves. The same codebase can be ready for ten internal users and unready for ten thousand customers.

3.1

The eight enterprise-readiness pillars

3.2

Maturity model

3.3

The decision gate



3.1 The eight enterprise-readiness pillars

This paper organizes enterprise readiness into eight pillars. The first four determine whether the application is built right. The last four determine whether it can be run right.

Eight pillars carry an enterprise-ready service



Figure 3: The Eight Pillars of Enterprise Readiness

1. Business and product fit:

A named business owner, a defined user group, a measurable value hypothesis, and a deliberate decision to invest.

2. Architecture and engineering quality:

A modular design with clear boundaries, documented decisions, and AI components that can be replaced without rewriting the application.

3. Security and privacy:

Strong identity, least-privilege access, managed secrets, secure coding practices, and privacy by design.

4. Data and AI governance:

Mapped data flows, classified data, versioned prompts and models, and evidence from evaluations.

5. Quality assurance and testing:

Automated tests at several levels, AI evaluation suites, and release gates that block regressions.

6. Reliability, resilience, and performance:

Service-level objectives, capacity planning, and recovery procedures that have been tested, not just written.

7. Operations, support, and observability:

Logs, metrics, traces, alerts, runbooks, and a clear path from a user's problem to a fix.

8. Compliance, risk, and auditability:

Evidence that controls exist and operate, a risk register, and a process for accepting residual risk.

No pillar is optional for an enterprise service. The maturity model below decides how deep each pillar must go at each stage.

3.2 Maturity model

Applying full enterprise controls to every experiment would kill the speed that makes vibe coding valuable. Applying none would let risk flow into production unchecked. A four-stage maturity model resolves the conflict by matching control depth to consequence.

Controls grow as an application earns its way to enterprise service

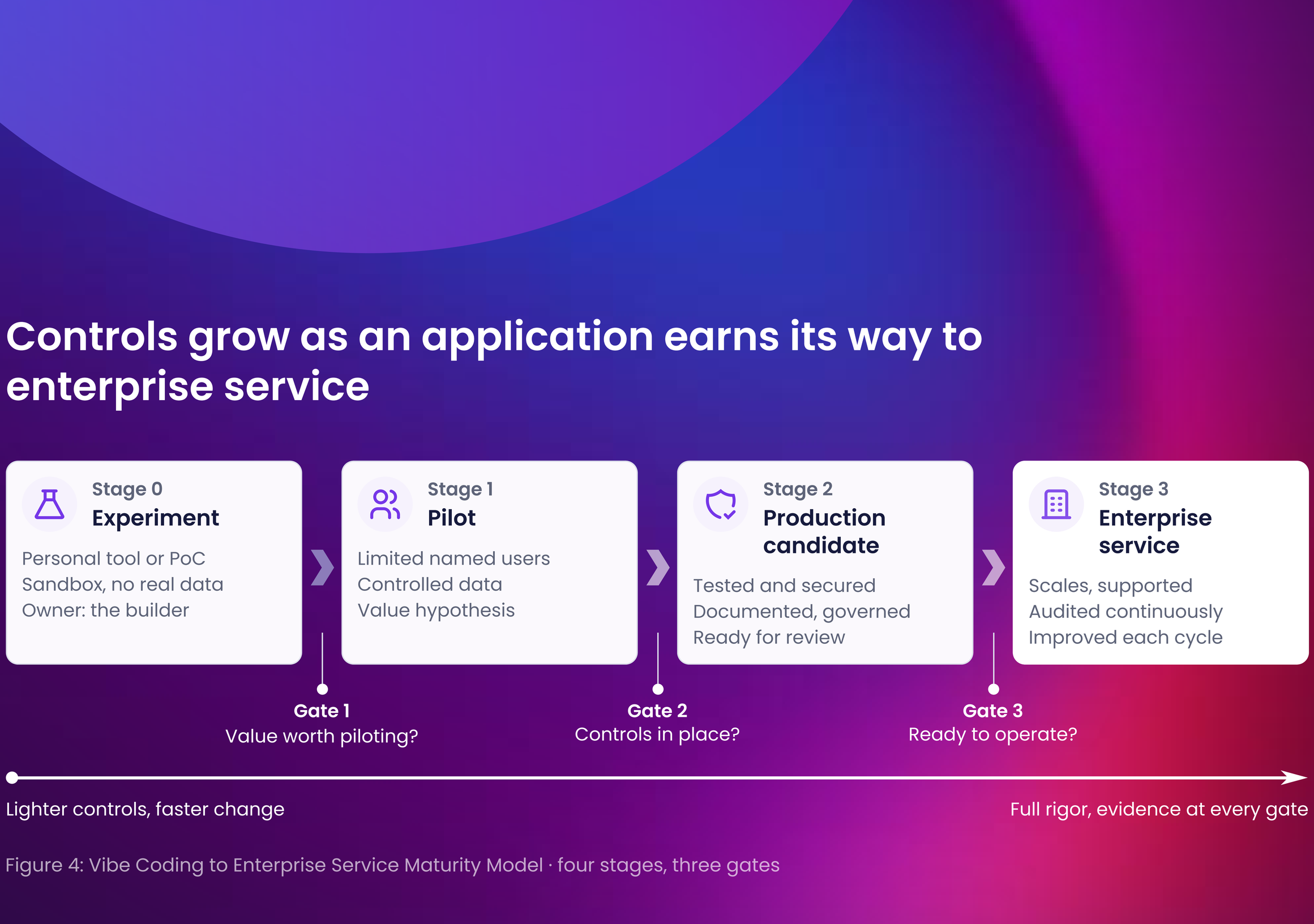


Figure 4: Vibe Coding to Enterprise Service Maturity Model · four stages, three gates

STAGE 0

Experiment.

A personal productivity tool or proof of concept. It runs in a sandbox with synthetic or public data. The only required controls are basic hygiene: no production credentials, no regulated data, and a clear label that it is experimental.

STAGE 1

Pilot.

A limited set of named users, controlled data, and a written value hypothesis. It needs an owner, source control, secrets removed from code, and basic logging.

STAGE 2

Production candidate.

Tested, secured, documented, and governed. It passes a threat model, has automated tests, runs through a pipeline, and has a named on-call path.

STAGE 3

Enterprise service.

Scalable, supportable, auditable, and continuously improved. It has SLOs, incident response, audit evidence, and a place in the organization's governance cadence.

> Each move between stages crosses a gate. The gates are where the organization spends its governance effort, rather than spreading it evenly across every idea.

3.3 The decision gate

At each gate, one question matters more than any checklist:

- **Experiment**
- **Pilot**
- **Production candidate**
- **Enterprise service**

THE GATE QUESTION

Is this application valuable enough to industrialize, or should it remain a controlled experiment?

Three answers are legitimate:

Advance.

The value is proven or strongly indicated, and the organization is willing to fund the controls the next stage requires.

Hold.

The application stays at its current stage, with its current constraints, until the value is clearer.

Retire.

The experiment taught what it needed to. The code is archived, credentials are revoked, and data is deleted.

KEY INSIGHT

Retiring is not failure. A healthy portfolio retires most of its experiments, and the discipline to do so frees investment for the few that deserve enterprise treatment.

04

The Transformation Framework

The framework moves an application through six phases, each with a clear objective and concrete deliverables. It is designed to preserve working code wherever possible: most vibe-coded applications need hardening, restructuring, and surrounding controls, not a ground-up rewrite.

4.1
Discover and triage

4.2
Stabilize the foundation

4.3
Secure the application and its AI components

4.4
Engineer for quality and change

4.5
Operationalize and scale

4.6
Govern and continuously improve

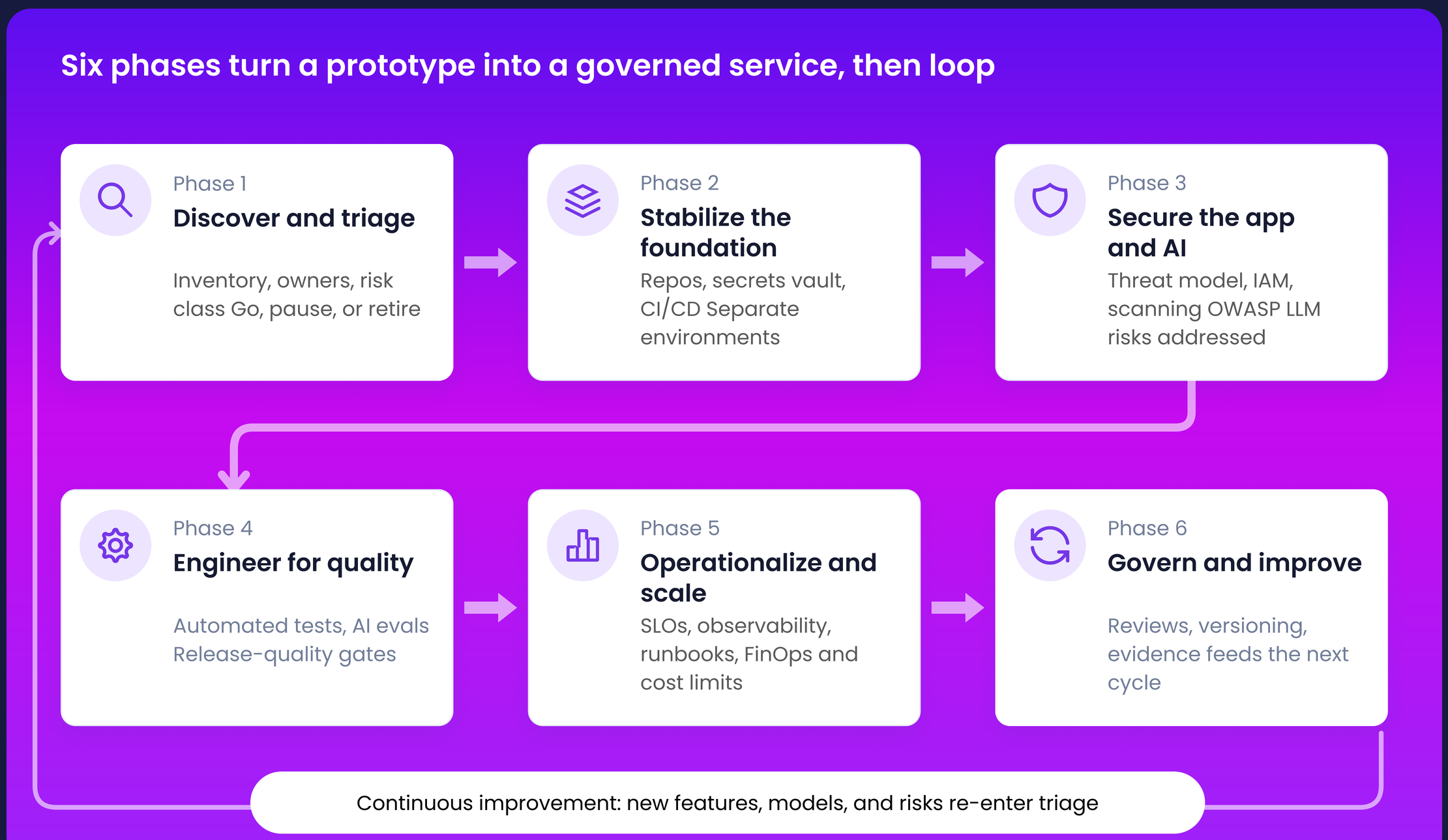


Figure 5: The Six-Phase Transformation Framework · a cycle, not a one-time project

The phases run in order for the first pass. After that, Phase 6 feeds the cycle: a new model version, a new integration, or a new regulation sends the relevant change back through triage rather than straight into production.

4.1 Phase 1: Discover and triage

Objective: Decide what exists, what it does, and whether it should move forward.

Vibe-coded applications often have no single source of truth. The first job is to build one:

- Inventory every repository, prompt, integration, model, API, dataset, credential, and deployment environment the application touches.
- Identify four owners: business, technical, data, and security. A missing owner is itself a finding.
- Document intended users and use cases, including the ones that emerged informally.
- Map sensitive and regulated data: personal data, financial records, health information, customer confidential data.
- Assess code origin and third-party dependencies, including which parts were AI-generated and never reviewed.
- Classify criticality as internal productivity, customer-facing, revenue-impacting, regulated, or safety-impacting.

Deliverables: application inventory, initial risk assessment, architecture snapshot, data-flow diagram, and a go, pause, or retire decision.

4.2 Phase 2: Stabilize the foundation

Objective: Convert an experimental codebase into a controlled engineering asset.

This phase makes the code safe to work on. It changes very little of what the application does:

- Apply source-control standards, with branching and pull-request workflows that require review before merge.
- Move every secret into a managed vault and rotate anything that was ever committed or shared.
- Separate development, test, staging, and production environments, each with its own credentials and data.
- Pin dependencies and generate a software bill of materials (SBOM).
- Express infrastructure as code so environments can be rebuilt, reviewed, and audited.
- Stand up a baseline CI/CD pipeline with code-quality gates: linting, static analysis, and a build that must pass.

Deliverables: repository standards, build pipeline, secrets vault integration, dependency inventory, and a deployment baseline.

4.3 Phase 3: Secure the application and its AI components

Objective: Apply security by design, not security as a final checklist.

Security work starts with a threat model: what is valuable, who might attack it, and how. From there, the team implements identity, authentication, and authorization; enforces least privilege; designs APIs securely; encrypts data in transit and at rest with managed keys; and adds vulnerability and supply-chain scanning to the pipeline.

AI components need their own attention. The OWASP Top 10 for LLM Applications is the most useful organizing reference. The risks that matter most for vibe-coded applications are:

- **Prompt injection:** instructions hidden in user input or retrieved content that redirect the model.
- **Sensitive-information disclosure:** the model revealing data the user should not see.
- **Excessive agency:** agents holding more permissions than their task needs.
- **Improper output handling:** model output passed unchecked into queries, code, or HTML.
- **Vector and embedding weaknesses:** retrieval stores that ignore document-level permissions.
- **Unbounded consumption:** runaway usage that becomes a cost or availability attack.

Deliverables: threat model, security requirements, access-control matrix, AI risk register, and a security test plan.

4.4 Phase 4: Engineer for quality and change

Objective: Make the application safe to modify repeatedly.

A vibe-coded application without tests freezes: every change risks a silent regression, so people stop changing it. This phase restores the ability to change with confidence:

- Agree a definition of done and coding standards that apply to human and AI-generated code alike.
- Build unit, integration, end-to-end, and regression tests, starting with the paths that carry the most business value.
- Add contract tests for APIs so integrations fail in the pipeline, not in production.
- Manage test data deliberately, using synthetic or masked data instead of production copies.
- Run performance and load tests against realistic volumes.
- Create AI evaluation suites that score quality, safety, hallucination, retrieval accuracy, and prompt-injection resistance on every prompt or model change.
- Design human-in-the-loop workflows for consequential decisions, such as approvals, payments, and customer communications.

Deliverables: test strategy, automated test suite, acceptance criteria, AI evaluation scorecard, and release-quality gates.

4.5 Phase 5: Operationalize and scale

Objective: Ensure the solution can be run, supported, measured, and improved.

Production software needs people and processes around it. This phase establishes observability through logs, metrics, traces, and audit logs; defines service-level indicators and objectives; writes incident-response procedures, runbooks, and escalation paths; and tests backup, disaster recovery, and business continuity.

It also covers capacity planning, release management with rollback procedures, customer support and feedback loops, and FinOps. AI features make cost management a first-class operational concern, because a single prompt change can multiply token usage.

Google's SRE guidance on service-level objectives offers the right balancing mechanism. An **SLO** and its **error budget** turn reliability into an explicit trade-off: when the budget is healthy, teams ship features; when it is spent, they invest in reliability.

Deliverables: SLO and SLA matrix, monitoring dashboards, incident-response plan, operational runbook, and a cost-management dashboard.

4.6 Phase 6: Govern and continuously improve

Objective: Keep the product enterprise-ready as software, models, data, and regulations evolve.

Enterprise readiness decays if nobody maintains it. Models are deprecated, dependencies acquire vulnerabilities, and regulations change. Phase 6 establishes a lightweight technical governance forum, periodic security and privacy reviews, version control for models and prompts, data retention and deletion policies, a risk-acceptance process, audit-evidence collection, vendor reviews, and a continuous-improvement backlog.

The NIST AI Risk Management Framework provides a practical lens with four functions.

Govern sets culture, roles, and policy.

Map establishes context and identifies risks.

Measure assesses and tracks them.

Manage prioritizes and acts on them.

Table 2: Transformation phases, owners, and deliverables

Phase	Lead owner	Key deliverables	Exit criterion
1. Discover and triage	Business owner with solution architect	Inventory, risk assessment, data-flow diagram	Go, pause, or retire decision recorded
2. Stabilize the foundation	Engineering lead	Repo standards, CI pipeline, vault, SBOM	No secrets in code; every change goes through the pipeline
3. Secure the application and AI	Security lead	Threat model, access matrix, AI risk register	High and critical risks mitigated or formally accepted
4. Engineer for quality	Engineering lead with QA	Test suite, AI evaluation scorecard, quality gates	Critical paths covered; evals pass agreed thresholds
5. Operationalize and scale	Platform or SRE owner	SLOs, dashboards, runbooks, cost controls	Production-readiness review passed
6. Govern and improve	Governance owner	Review cadence, version history, audit evidence	Evidence current at each quarterly review

05

Reference Architecture for Enterprise-Ready AI Applications



The most common architectural flaw in vibe-coded AI applications is that everything talks to everything. The user interface calls the model directly, the model call contains business rules, and credentials sit wherever they were first needed. A reference architecture restores boundaries, so each concern can be secured, tested, and replaced on its own.

5.1 Security

5.2 Layered architecture

5.3 Governance

5.1 Architecture principles

Eight principles guide the target design:

Modular design

Components have one responsibility and a defined interface.

Clear separation of concerns

Presentation, business logic, integration, AI orchestration, and data live in distinct layers.

API-first integration

Systems connect through documented, versioned APIs rather than shared databases or scraped screens.

Zero-trust principles

Every request is authenticated and authorized, including service-to-service calls inside the network.

Secure defaults

New endpoints, storage buckets, and tools start closed and are opened deliberately.

Observable by design

Every layer emits logs, metrics, and traces with a shared correlation ID.

Replaceable AI model and vendor layers

No business logic depends on one provider's model or SDK.

Controlled autonomy for agents

Agents act only through scoped tools, within budgets, and under approval rules.

5.2 Architecture layers

Figure 6 shows the eight layers. Six stack in the order a request travels. Security and governance cut across all of them.

Security

- IAM and SSO
- Secrets vault
- Encryption
- Audit logs

Experience layer

Web, mobile, chat, and internal tools

↓

Application layer

Business logic, workflows, policy enforcement

↓

Integration layer

APIs, event streams, connectors to systems of record

↓

AI orchestration layer

Prompt registry, model gateway, routing, guardrails, evals

↓

Data layer

Transactional DBs, warehouse, vector stores, document repos

↓

Platform layer

Cloud, containers, CI/CD, observability

Governance

- Policies
- Controls
- Approvals
- Evidence

Experience layer: web, mobile, chat, and internal tools. It handles presentation and session management, never business rules or model calls.

Application layer: business logic, workflows, and policy enforcement. This is where "only managers can approve discounts above 15%" lives, in code that can be tested.

Integration layer: APIs, event streams, and connectors to systems of record such as ERP, CRM, and HR platforms.

AI orchestration layer: the prompt registry, model gateway, routing, guardrails, and evaluation hooks. Every model call passes through here, which makes it the single place to enforce limits, log usage, and switch providers.

Data layer: transactional databases, data warehouses, vector stores, and document repositories, each with access control that reflects the source data's permissions.

Platform layer: cloud infrastructure, containers, CI/CD, and observability tooling.

Security layer: identity and access management, secrets, encryption, and audit logging, applied at every layer.

Governance layer: policies, controls, approvals, and the evidence that proves they operate.

The model gateway deserves emphasis. In a typical vibe-coded application, model calls are scattered across the codebase with a hard-coded API key. Consolidating them behind a gateway is often the single highest-value architectural change, because it delivers cost control, observability, provider independence, and a policy enforcement point at once.

5.3 Design decisions to document

AI applications involve decisions that conventional architecture documents rarely capture. Each of the following should be written down, reviewed, and revisited when circumstances change:

Why a specific model or provider was selected, and what would trigger a switch.

What data can and cannot be sent to models, and under what contractual terms.

How retrieval-augmented generation (RAG) respects document permissions.

What actions agents may take, with what limits.

Where human approval is mandatory.

How the application behaves when a model is slow, unavailable, or returns poor output, including any degraded mode.

Sidebar: Architecture Decision Records (ADRs)

Vibe-coded applications are full of decisions that were never consciously made. The model was chosen because it was the default. The vector database was chosen because a tutorial used it. An ADR turns each of those informal choices into a durable engineering decision.

An ADR is a short document, usually a page, that records the context, the decision, the alternatives considered, and the consequences. ADRs live in the repository beside the code. When a new engineer asks "why do we do it this way?", the answer exists, and it includes the reasoning. **Appendix C provides a template.**

06

Security, Privacy, and Responsible AI Controls



Security for a vibe-coded application has two layers. The first is conventional application security, which AI-generated code often skips. The second is AI-specific security, which most organizations are still learning. Both must be in place before an application reaches Stage 2.

6.1 **Secure SDLC baseline**

6.2 **AI-specific security controls**

6.3 **Privacy by design**

6.4 **Responsible AI controls**

6.1 Secure SDLC baseline

The NIST Secure Software Development Framework (SP 800-218) describes core practices that integrate into any development lifecycle to reduce vulnerabilities and improve response readiness. Its four practice groups map cleanly onto vibe-coded remediation:

Prepare the organization (PO)

Define security requirements, roles, and toolchains, including which AI coding assistants are approved and how their output is reviewed.

Protect the software (PS)

Protect code, build artifacts, and release integrity from tampering. Signed commits, protected branches, and provenance for builds apply equally to AI-generated code.

Produce well-secured software (PW)

Threat modeling, secure design, code review, and automated testing. The key adjustment for AI-generated code is that review must be real: an approving click on a thousand-line pull request is not review.

Respond to vulnerabilities (RV)

Identify, analyze, and remediate vulnerabilities, and find the root cause so the same flaw is not regenerated tomorrow.

That last point matters more than it first appears. AI assistants tend to reproduce the same insecure pattern wherever they are asked for similar code. Fixing one instance without updating team guidance, linters, or prompt templates invites a repeat.

6.2 AI-specific security controls

AI features add an attack surface that conventional scanners do not see. Eight controls address it:

Treat prompts as managed artifacts. Store system prompts in version control, review changes, and test them before release.

Validate and constrain model outputs. Enforce schemas, strip unexpected markup, and never pass raw output into SQL, shell commands, or HTML.

Apply fine-grained authorization to tools and agents. An agent inherits the permissions of the user it serves, never broader service-account rights.

Isolate user sessions and tenants. Conversation memory, caches, and retrieval results must never cross users or customers.

Implement rate limits, quotas, and budget limits. Cap usage per user, per feature, and per day, and alert well before the cap.

Scan retrieval sources and external content. Treat documents, web pages, and tickets as untrusted input that may carry hidden instructions.

Maintain red-team and adversarial testing. Run prompt-injection and jailbreak test suites on every prompt or model change.

Prevent secret leakage and sensitive-data exposure. Keep credentials out of prompts entirely and filter outputs for personal and confidential data.

Figure 7 shows how these controls combine on a single request.

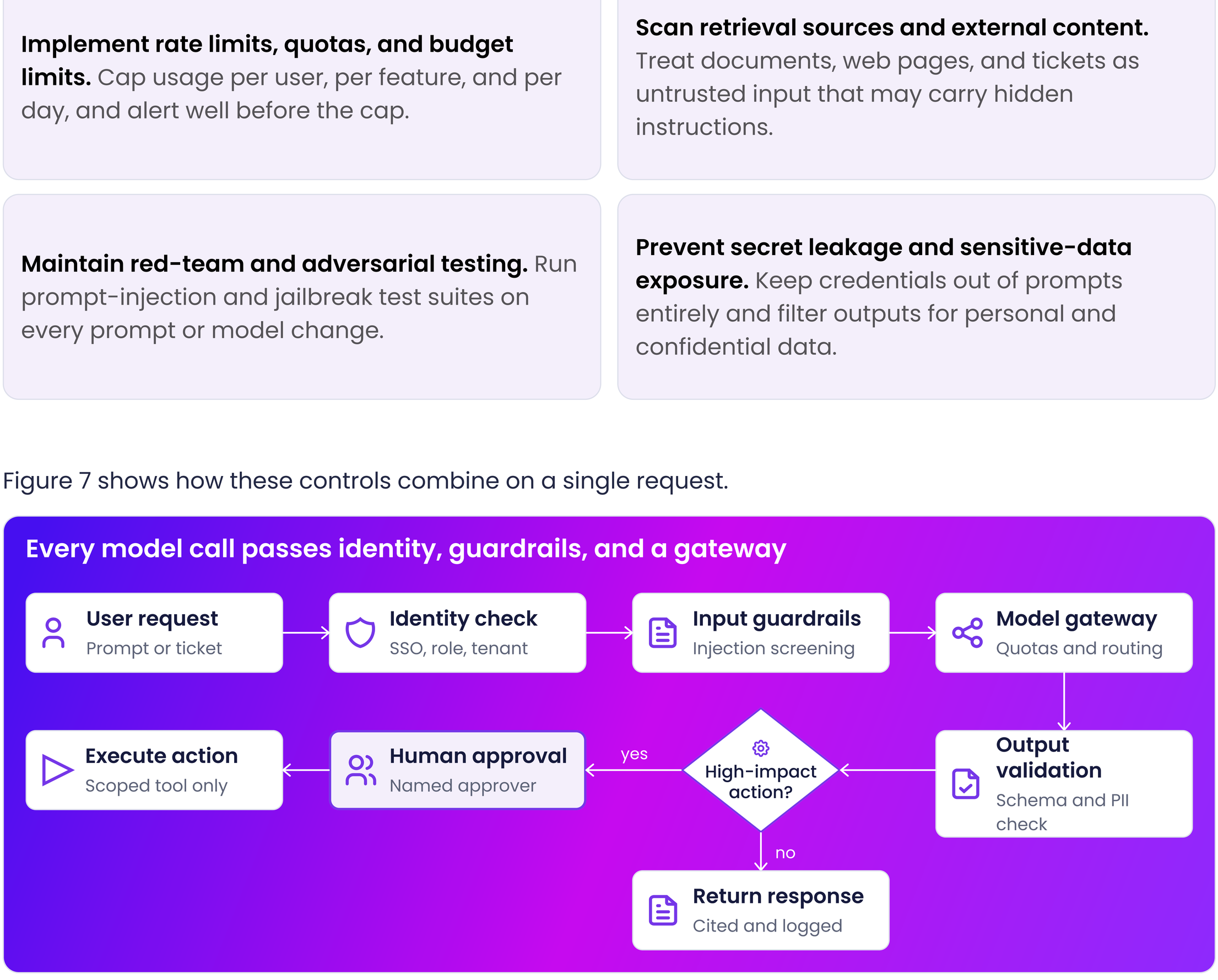


Figure 7: Guarded Request Path for an AI Feature · one decision, one human gate

A request is authenticated and screened before it reaches the model gateway. Output is validated before anyone sees it, and any high-impact action waits for a named human approver.

6.3 Privacy by design

Privacy failures in vibe-coded applications are rarely malicious. They happen because a prompt was the quickest way to pass context, or because a log statement captured a full request body. Privacy by design makes the safe path the easy one:

Data minimization:
send models only the fields a task requires.

Purpose limitation:
use data only for the purpose users were told about.

Consent and disclosure:
tell users when AI processes their data and when they are interacting with AI.

Data classification:
label data by sensitivity so controls can follow the label.

Retention and deletion:
define how long prompts, outputs, and logs are kept, and delete them on schedule.

Cross-border considerations:
know where each model provider processes and stores data.

PII masking and tokenization:
replace identifiers before data leaves your boundary where possible.

Audit trails:
record who accessed sensitive data and which AI actions touched it.

6.4 Responsible AI controls

Responsible AI is not a separate program bolted on at the end. It is a set of product controls sized to the use case:

Explainability appropriate to the use case:
A drafting assistant needs little; a system that influences credit, hiring, or medical decisions needs much more.

Human oversight:
People review and can override consequential outputs.

Fairness and bias testing where applicable:
Test outcomes across relevant user groups when the application affects people differently.

Clear user disclosures:
Users know what the AI does, what it cannot do, and how to reach a person.

Monitoring for harmful or inaccurate outputs:
Production sampling and user feedback feed the evaluation suite.

Escalation and correction processes:
A reported error has an owner, a fix path, and a way to correct any affected records.

Table 3: AI risk to control mapping

The mapping below links the most common AI risks, drawn from the OWASP Top 10 for LLM Applications, to the controls that address them first.

Risk	Example	Primary controls
Prompt injection	A malicious document changes an agent's behavior	Input isolation, tool permissions, adversarial tests
Sensitive-data leakage	A user receives another customer's data	Tenant isolation, access control, output filtering
Excessive agency	An agent sends an unauthorized email or payment	Least privilege, approval gates, action logs
Improper output handling	Model output is executed as a database query	Schema validation, parameterized queries, encoding
Vector-store weakness	Retrieval returns documents the user may not read	Permission-aware retrieval, document classification
Misinformation	The AI gives incorrect policy guidance	Grounded retrieval, citations, human review
Unbounded consumption	Model calls create unexpected cost	Usage limits, model routing, budget alerts
Supply-chain compromise	A tampered model or package enters the build	SBOM, provenance checks, pinned versions

07

Case Studies: Lessons from the Field

The three case studies below are anonymized composites, built from patterns that recur across AI-assisted builds. They are illustrative, not accounts of specific clients, and they do not report client-specific figures. Each follows the same format: business problem, vibe-coded starting point, risks identified, transformation decisions, controls implemented, outcomes to measure, and lessons.



7.1 Internal Knowledge Assistant

7.2 Customer-Facing AI Workflow Agent

7.3 Legacy Workflow Modernization

Case Study 1:

Internal knowledge assistant

Business problem.

Employees at a mid-sized financial-services firm spent significant time searching shared drives for current HR, travel, and compliance policies. They often found outdated versions.

Vibe-coded starting point.

A people-operations analyst built a chat assistant over a weekend. It indexed every document in three shared drives into a vector store and answered questions through a popular model API. Within six weeks, several hundred employees were using it daily.

Enterprise risks identified.

No document-level permissions: the index included board papers and draft restructuring plans, which any user could surface with the right question.

No citation validation: answers blended current and superseded policies without saying which was which.

No logging of sensitive queries, so nobody could tell what had been exposed.

A single shared model API key, billed to the analyst's team card.

Transformation decisions.

The firm chose to keep the working interface and retrieval approach, and to rebuild the ingestion and access layer around them. A rewrite was rejected because the user experience was already validated.

Internal knowledge assistant

Controls, outcomes and lesson

Architecture and controls implemented.

SSO integration and role-based retrieval that checks the source document's permissions for every result.

Document classification at ingestion, with restricted and draft documents excluded by default.

Prompt and model versioning through a model gateway, with the shared key replaced by scoped credentials.

An evaluation dataset of real policy questions with approved answers, run on every change.

Audit logging of queries and retrieved documents, with retention aligned to policy.

Guardrails for unsupported answers: when retrieval confidence is low, the assistant says it does not know and links to the policy owner.

Outcomes to measure.

Unsupported-response rate, share of answers with a valid citation to a current policy, access-control exceptions found in audit, and average time to find approved policy information.

Lesson for readers.

The riskiest part of a RAG assistant is rarely the model. It is the retrieval layer that decides what the model is allowed to see.

Case Study 2:

Customer-facing AI workflow agent

Business problem.

A B2B SaaS startup wanted to resolve routine support requests, such as plan changes, seat additions, and invoice copies, without adding support headcount.

Vibe-coded starting point.

Two founders built an agent that read incoming support tickets, decided what the customer wanted, and called the billing and account APIs directly. Early results were strong, and the agent resolved many tickets end to end.

Enterprise risks identified.

Excessive API permissions

The agent used an admin token that could modify any account.

Prompt injection through ticket content

A test ticket containing hidden instructions persuaded the agent to apply a discount.

No human approval for high-impact actions such as refunds and plan downgrades.

Cost volatility

Retries and long ticket threads created unpredictable model spend.

Transformation decisions.

The founders kept the agent but narrowed what it could do, and when. Their first enterprise prospect's security questionnaire made the work urgent.

Customer-facing AI workflow agent

Controls, outcomes and lesson

Architecture and controls implemented.

Scoped tools: each action became a narrow tool with its own permission, limited to the requesting customer's account.

An approval workflow: refunds, downgrades, and anything above a defined value route to a named support lead.

Prompt-injection testing in the pipeline, using a growing library of adversarial tickets.

Rate limits and cost budgets per customer and per day, with alerts at 80% of budget.

Incident playbooks for wrong actions, including how to reverse them and notify the customer.

SLOs for first-response time and successful task completion, reviewed weekly.

Outcomes to measure.

Unsafe-action prevention rate, human escalation rate, successful task completion, cost per resolved ticket, and time to pass enterprise security review.

Lesson for readers.

An agent's risk is the product of its permissions and its gullibility. Reduce both, and put a human in front of anything that is hard to undo.

Case Study 3:

03 Legacy workflow modernization

- **Business problem:**

  A manufacturing enterprise ran capital-expenditure approvals on a fifteen-year-old internal application that few engineers still understood.

- **Vibe-coded starting point:**

  An engineering team used AI coding assistants to generate a modern web front end and services that replicated the legacy workflow. Generation was fast, but the team discovered that the legacy system encoded approval rules nobody had documented.

- **Enterprise risks identified:**

  Undocumented business rules, direct database writes that bypassed audit trails, no tests to confirm the new system matched the old, and a planned "big bang" cutover.

Legacy workflow modernization

Transformation, outcomes and lesson

Transformation focus.

Domain-driven decomposition to separate approval rules, notifications, and reporting into distinct services.

An API layer around the legacy system, so old and new could run side by side.

Test automation before refactoring: characterization tests captured the legacy system's actual behavior, including its quirks.

Progressive rollout by business unit, starting with the lowest-value approvals.

Observability and rollback so any unit could return to the legacy path within minutes.

A change-management strategy that trained approvers and published the rules now explicit in code.

Outcomes to measure.

Parity defects found before versus after rollout, approval cycle time, audit exceptions, and rollback events.

Lesson for readers.

AI accelerates writing the new system. It does not tell you what the old one really did. Characterization tests are the cheapest insurance in any modernization.

08

Operating Model: People, Process, and Governance



Technology controls fail without clear accountability. The most common reason a vibe-coded application stalls on its way to production is not a missing tool. It is that no one has the authority, and the time, to make the decisions each phase requires.

8.1
Roles and accountabilities

8.2
Two-speed delivery model

8.3
Governance gates

8.1 Required roles

Small organizations will combine roles, and one person may hold several. What matters is that each accountability has a name against it.

Table 4 · Roles and accountabilities

Role	Accountable for	Must sign off at
Business owner	Value hypothesis, funding, advance or retire decision	Gate 1 and every quarterly review
Product manager	Requirements, acceptance criteria, user feedback loop	Gate 1 and release planning
Solution architect	Target architecture, ADRs, integration design	Gate 2
Engineering lead	Code quality, pipeline, tests, technical debt	Gates 2 and 3
Security lead	Threat model, security requirements, risk acceptance advice	Gate 2
Data and AI governance owner	Data classification, model and prompt governance, evaluations	Gates 2 and 3
Platform or DevOps owner	Environments, observability, cost controls	Gate 3
Legal, privacy, and compliance	Data-processing terms, disclosures, regulatory fit	Gate 2 for regulated data
Support and operations owner	Runbooks, on-call, incident response, user support	Gate 3

8.2 The two-speed delivery model

The answer to the tension in Section 1 is not a single process. It is two lanes with a clear boundary between them.

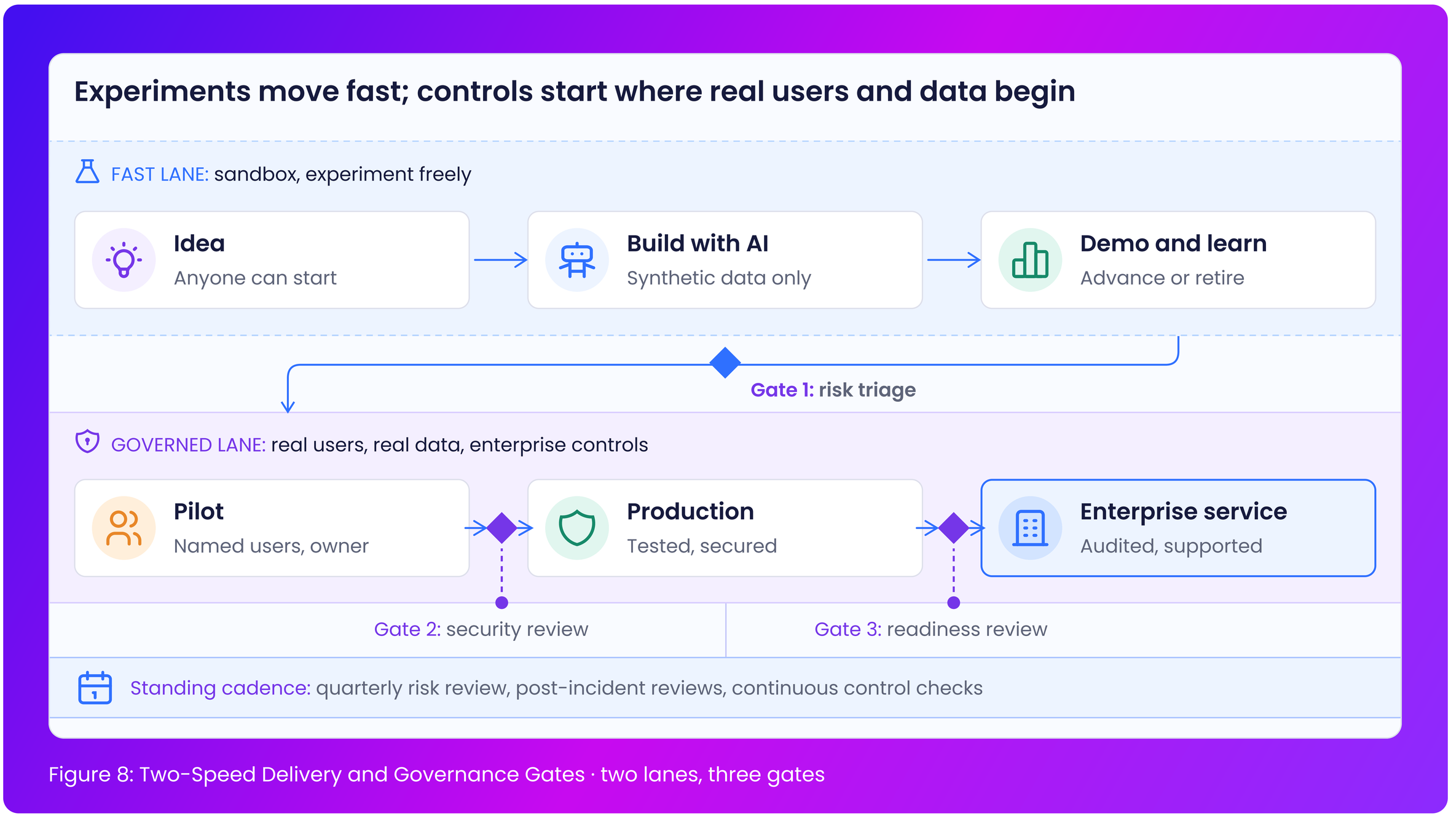


Figure 8: Two-Speed Delivery and Governance Gates · two lanes, three gates

Preserve speed in the fast lane. Anyone can experiment in a sandbox with approved AI tools, synthetic or public data, and no production credentials. No review board is involved.

Enforce controls in the governed lane. The moment an application touches real users or real data, it crosses Gate 1 and the controls of the maturity model apply.

Avoid heavyweight governance for every idea. Governance effort concentrates at the gates, not across the whole portfolio.

Use risk-based thresholds. An internal tool with non-sensitive data may pass Gate 2 with a lightweight review. A customer-facing agent that moves money needs the full treatment.

The boundary must be enforced technically, not just by policy. Sandbox environments should be unable to reach production data stores, and production credentials should be unavailable to sandbox tools.

8.3 Governance cadence

Governance works when it is predictable. A recommended cadence:

Architecture review at key maturity gates. Short, focused sessions using the ADRs and diagrams produced in earlier phases.

Security review before production. Threat model, test results, and open risks reviewed by the security lead.

Model and data reviews for AI capabilities. Evaluation results, data flows, and any change of model or provider.

Quarterly risk review. The risk register, cost trends, incidents, and the portfolio's advance, hold, and retire decisions.

Post-incident reviews. Blameless, focused on system fixes, with actions tracked to completion.

Continuous control validation. Automated checks that controls still operate, such as secrets scanning, dependency alerts, and access reviews.

09

Metrics That Prove Enterprise Readiness

Enterprise readiness is a claim, and claims need evidence. Four families of metrics provide it. Measure a baseline in Phase 1, even if the numbers are uncomfortable, so the transformation can show progress rather than assert it.

9.1 **Engineering metrics**

9.2 **Reliability metrics**

9.3 **AI quality metrics**

9.4 **Business metrics**

EVIDENCE, NOT ASSERTION

Metrics That Prove Enterprise Readiness

9.1 Engineering metrics

DORA's delivery metrics remain the best-tested starting point: **deployment frequency**, **lead time for changes**, **change failure rate**, and **time to restore service**.

Add three that matter especially for AI-generated code: **test coverage of critical paths**, **defect escape rate** (defects found in production versus before release), and **vulnerability remediation time** for dependencies.

9.2 Reliability metrics

Availability, **latency**, **error rate**, and **recovery time** describe the user's experience. **SLO compliance** and **error-budget consumption** turn those numbers into decisions about when to ship features and when to invest in stability.

9.3 AI quality metrics

AI features need metrics that conventional monitoring does not provide: **grounded-answer rate**, **unsupported-answer (hallucination) rate**, **prompt-injection resistance**, **retrieval relevance**, **human escalation rate**, and **unsafe-action prevention rate**.

Most are measured through the evaluation suite on every release, then sampled in production.

9.4 Business metrics

The transformation must pay for itself. Track **user adoption**, **task-completion rate**, **productivity improvement**, **cost per successful transaction**, **support-ticket reduction**, and **time to value** for new capabilities.

TABLE 5 · ENTERPRISE-READINESS METRICS AND STARTING TARGETS

Measure the evidence that makes change safe

Metric	Family	How to measure	Illustrative starting target
Change failure rate	Engineering	Share of deployments causing incidents or rollbacks	Trending down quarter over quarter
Time to restore service	Engineering	Median time from alert to recovery	Under one working day, then hours
Critical-path test coverage	Engineering	Share of critical user journeys with automated tests	All critical journeys covered
SLO compliance	Reliability	Share of windows meeting each SLO	SLOs met in most windows; breaches reviewed
Grounded-answer rate	AI quality	Eval-suite answers supported by cited sources	Agreed threshold per use case, gated in CI
Prompt-injection resistance	AI quality	Share of adversarial tests safely handled	No high-impact action triggered by any test
Human escalation rate	AI quality	Share of requests routed to people	Stable and explainable, not zero
Cost per successful transaction	Business	Model plus infrastructure cost divided by completed tasks	Known, budgeted, and alerted

Two cautions apply

First, avoid optimizing a single metric: an assistant can raise its grounded-answer rate by refusing more often. Second, never treat a zero human escalation rate as success in consequential workflows. It usually means escalation is broken.

10

Enterprise-Readiness Assessment Checklist

The scorecard below gives leaders a fast, honest view of where an application stands. It covers the ten control areas that most often separate a prototype from a production service, grouped under the eight pillars. Appendix A expands it into a full questionnaire.

10.1 How to use the scorecard

10.2 Enterprise-readiness scorecard

10.1 How to use the scorecard

A fast, honest view of where an application stands

The scorecard below gives leaders a fast, honest view of where an application stands. It covers the ten control areas that most often separate a prototype from a production service, grouped under the eight pillars. Appendix A expands it into a full questionnaire.

Not started
no work has begun, or the control exists only as an intention.

In progress
work is under way but not yet in effect.

Implemented
the control is in place and operating.

Verified
someone other than the implementer has checked it works, and evidence is stored.

Verification is the gate

Only “Verified” counts toward a gate. An application moving from Stage 1 to Stage 2 should have every control below at least Implemented, and the security and data controls Verified.

10.2 Enterprise-readiness scorecard

Table 6 · Enterprise-readiness scorecard

Control area	Control	Status
Business and ownership	Named business, technical, data, and security owners	[Status]
Business and ownership	Value hypothesis and success measures documented	[Status]
Architecture and maintainability	Architecture and data-flow diagrams are current	[Status]
Architecture and maintainability	ADRs recorded for model, data, and agent choices	[Status]
Secure development	Source repository with protected branches and review	[Status]
Secure development	Secrets removed from code and rotated	[Status]
Secure development	Threat model completed and reviewed	[Status]
Identity and access management	SSO, MFA, and role-based access in place	[Status]

10.2 Enterprise-readiness scorecard — continued

TABLE 6 · ENTERPRISE-READINESS SCORECARD

Controls, evidence, and current status

Control area · continued	Control	Status
Data protection and privacy	Data classified; retention and deletion defined	[Status]
AI model and prompt governance	Prompts and model versions under version control	[Status]
AI model and prompt governance	AI evaluation suite implemented and gated in CI	[Status]
Testing and release readiness	Automated tests cover critical user journeys	[Status]
Reliability and observability	SLOs and alerts configured	[Status]
Incident response	Runbook and on-call path tested	[Status]
Compliance and evidence	Risk register current, with owners and review dates	[Status]
Compliance and evidence	Audit evidence collected and stored	[Status]

Turn gaps into the transformation backlog

A scorecard with many “Not started” entries is not a reason to stop.
It is the backlog for the 90-day path in the next section.

11

Recommended 90-Day Transformation Path

Ninety days is enough to take a valuable vibe-coded application from Stage 1 to a controlled production rollout, provided the owners are named in the first week. The plan below assumes a small, dedicated team and an application of moderate complexity. Regulated or customer-facing applications with agents may need longer in the second phase.

Days 1–30

Assess and stabilize

11.1

Days 31–60

Secure and test

11.2

Days 61–90

Operationalize

11.3



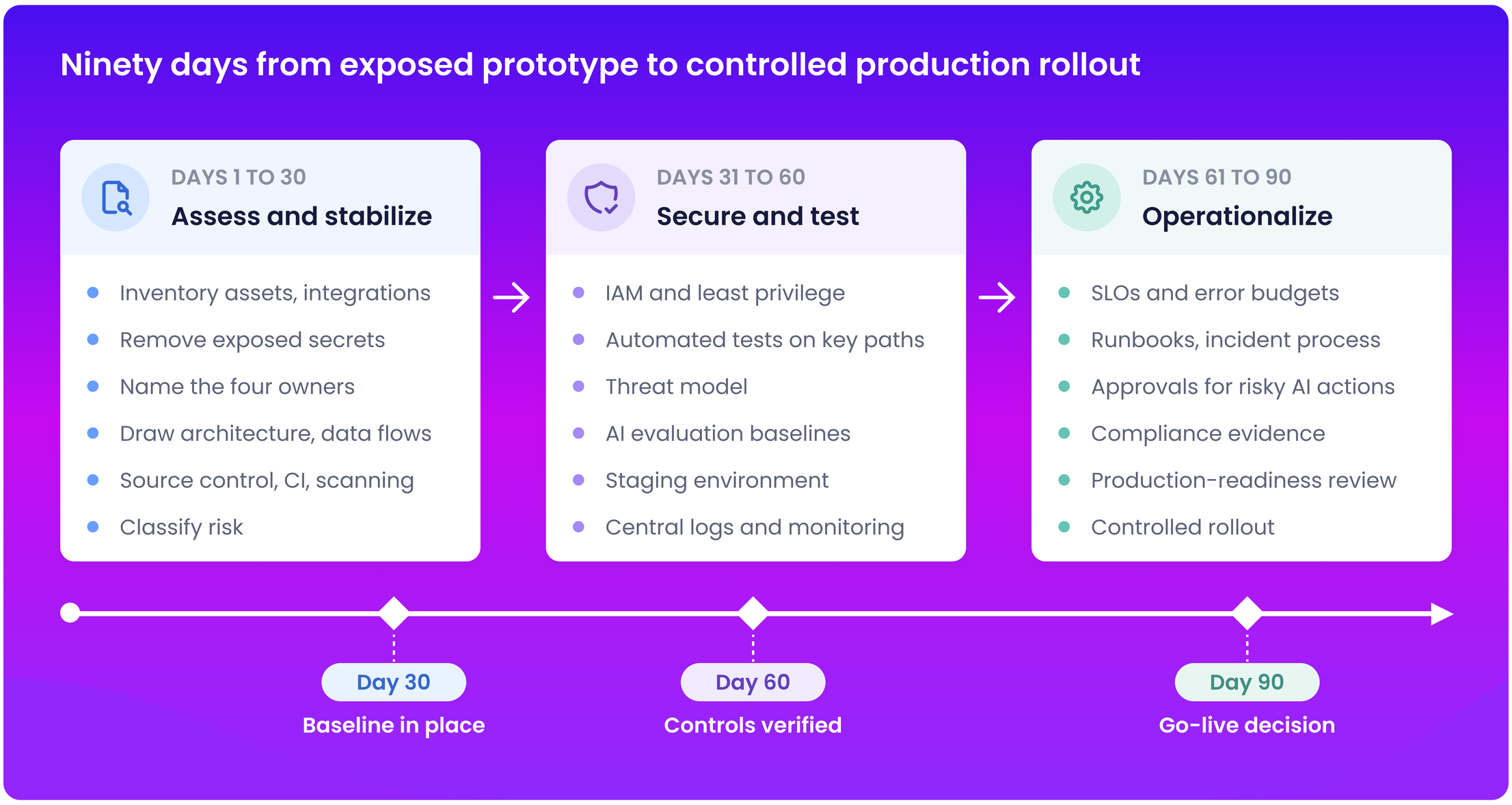


Figure 9: The 90-Day Transformation Roadmap · three 30-day phases, three gates

11.1 Days 1–30: Assess and stabilize

The first month removes the most urgent risks and creates a baseline to improve from:

- Inventory assets, integrations, models, and data stores.
- Remove exposed secrets, move them to a vault, and rotate them.
- Establish the business, technical, data, and security owners.
- Create architecture and data-flow diagrams.
- Set up source control standards, CI, and basic dependency and secrets scanning.
- Classify the application's risk and record the go, pause, or retire decision.

Day 30 gate: baseline in place, secrets rotated, and a written decision to proceed.

11.2 Days 31–60: Secure and test

The second month builds the controls that allow the application to change safely:

- Implement IAM with SSO and least privilege, including for agents and service accounts.
- Add test automation, starting with critical user journeys.
- Conduct threat modeling, including AI-specific threats.
- Establish AI evaluation baselines for quality, safety, and injection resistance.
- Set up a staging environment that mirrors production.
- Add centralized logs and monitoring.

Day 60 gate: high and critical risks mitigated or formally accepted, and tests and evals running in the pipeline.

11.3 Days 61–90: Operationalize

The third month prepares the organization to run the service:

- Define SLOs and error budgets with the business owner.
- Complete runbooks and incident procedures, and run one incident drill.
- Implement approval workflows for high-risk AI actions.
- Finalize compliance evidence and update the risk register.
- Conduct a production-readiness review against the scorecard in Section 10.
- Launch a controlled production rollout, by user group or region, with rollback ready.

Day 90 gate: a go-live decision made by the business owner, with evidence rather than optimism.

Conclusion

Vibe coding is a powerful entry point, not a destination.

It has made the first version of software cheaper and faster than ever, and organizations that suppress it will lose ideas, talent, and time to competitors that do not.

But the first version was never the expensive part. The expensive part is everything that lets an organization depend on software: knowing who owns it, what it touches, how it fails, and how to prove it behaves. That work does not happen by accident, and AI does not do it by default. Enterprise readiness comes from intentional systems engineering.

The path this paper describes is deliberately incremental. Keep the fast lane open for experimentation. Put a clear gate where real users and real data begin. Move the applications that earn it through six phases of stabilization, security, quality, operations, and governance, and retire the rest without regret.

The winning model combines AI-enabled speed with secure architecture, accountable ownership, evidence-based governance, and operational rigor. Neither half is enough alone.

Where to start

Begin with a portfolio assessment of the AI-assisted applications already in use across your organization. Most leaders find more than they expected. Classify each by risk and value, apply the scorecard in Section 10, and use the framework to decide which applications deserve enterprise investment, which should stay controlled experiments, and which should be retired.

The applications that matter most to your business are probably already running.
The question is whether they are running on purpose.

Appendices · Appendix A · 1 / 3

Enterprise-readiness assessment questionnaire

Answer each question yes, partly, or no. Any "no" in the security, data, or AI governance pillars blocks Gate 2 unless the risk is formally accepted.

Business and product fit

6 questions

01	Is there a named business owner with budget authority?	○ Yes ○ Partly ○ No
02	Is the value hypothesis written down with a measurable success criterion?	○ Yes ○ Partly ○ No
03	Are the intended users and use cases documented?	○ Yes ○ Partly ○ No
04	Has anyone decided, explicitly, that this application should advance?	○ Yes ○ Partly ○ No
05	Is there a plan to retire the application if the hypothesis fails?	○ Yes ○ Partly ○ No
06	Are informal or unintended uses known?	○ Yes ○ Partly ○ No

Architecture and engineering quality

6 questions

07	Is there a current architecture diagram?	○ Yes ○ Partly ○ No
08	Are presentation, business logic, integration, and AI calls separated?	○ Yes ○ Partly ○ No
09	Do all model calls pass through a single gateway or abstraction?	○ Yes ○ Partly ○ No
10	Are key decisions recorded as ADRs?	○ Yes ○ Partly ○ No
11	Could another team understand and change the code within a week?	○ Yes ○ Partly ○ No
12	Is infrastructure defined as code?	○ Yes ○ Partly ○ No

Security and privacy

6 questions

13	Are all secrets in a managed vault and rotated?	○ Yes ○ Partly ○ No
14	Is there a current threat model?	○ Yes ○ Partly ○ No
15	Is access controlled through SSO with role-based permissions?	○ Yes ○ Partly ○ No
16	Do agents and service accounts follow least privilege?	○ Yes ○ Partly ○ No
17	Are dependencies scanned, with an SBOM generated?	○ Yes ○ Partly ○ No
18	Is data encrypted in transit and at rest?	○ Yes ○ Partly ○ No

Appendix A · 2 / 3

Enterprise-readiness assessment questionnaire

Data and AI governance

6 questions

19	Is all data the application touches classified?	○ Yes	○ Partly	○ No
20	Is there a data-flow diagram showing what reaches each model provider?	○ Yes	○ Partly	○ No
21	Are prompts and model versions under version control?	○ Yes	○ Partly	○ No
22	Does retrieval respect source-document permissions?	○ Yes	○ Partly	○ No
23	Are retention and deletion rules defined for prompts, outputs, and logs?	○ Yes	○ Partly	○ No
24	Are provider data-processing terms reviewed and signed?	○ Yes	○ Partly	○ No

Quality assurance and testing

6 questions

25	Do automated tests cover critical user journeys?	○ Yes	○ Partly	○ No
26	Do tests run on every change, and block release when they fail?	○ Yes	○ Partly	○ No
27	Is there an AI evaluation suite with agreed thresholds?	○ Yes	○ Partly	○ No
28	Are prompt-injection tests part of the suite?	○ Yes	○ Partly	○ No
29	Is test data synthetic or masked?	○ Yes	○ Partly	○ No
30	Has the application been load-tested at realistic volumes?	○ Yes	○ Partly	○ No

Reliability, resilience, and performance

6 questions

31	Are SLOs defined and agreed with the business owner?	○ Yes	○ Partly	○ No
32	Is there a tested backup and restore procedure?	○ Yes	○ Partly	○ No
33	Is there a degraded mode when a model provider fails?	○ Yes	○ Partly	○ No
34	Can a release be rolled back within minutes?	○ Yes	○ Partly	○ No
35	Is capacity planned for peak periods?	○ Yes	○ Partly	○ No
36	Is recovery time measured?	○ Yes	○ Partly	○ No

Evidence / notes

Appendix A · 3 / 3

Enterprise-readiness assessment questionnaire

Operations, support, and observability

6 questions

37	Are logs, metrics, and traces centralized and correlated?	○ Yes ○ Partly ○ No
38	Do alerts reach a named on-call person?	○ Yes ○ Partly ○ No
39	Is there a runbook for the most likely failures?	○ Yes ○ Partly ○ No
40	Is model and cloud cost tracked, budgeted, and alerted?	○ Yes ○ Partly ○ No
41	Do users know how to get support?	○ Yes ○ Partly ○ No
42	Has an incident drill been run?	○ Yes ○ Partly ○ No

Compliance, risk, and auditability

6 questions

43	Is there a current risk register with owners?	○ Yes ○ Partly ○ No
44	Is residual risk formally accepted by an accountable person?	○ Yes ○ Partly ○ No
45	Is audit evidence collected automatically where possible?	○ Yes ○ Partly ○ No
46	Are user-facing AI disclosures in place?	○ Yes ○ Partly ○ No
47	Have third-party vendors been reviewed?	○ Yes ○ Partly ○ No
48	Is the application on the quarterly governance agenda?	○ Yes ○ Partly ○ No

Evidence / notes

Appendix B

Sample risk register

Table 7: Sample risk register

Risk ID	Description	Asset or system	Likelihood	Impact	Rating	Owner	Residual risk	Review	Mitigation
R-01	Prompt injection via retrieved documents triggers an unintended action	Support agent	Medium	High	High	Security lead	Low	Quarterly	Scoped tools, approval gate, adversarial tests
R-02	Retrieval exposes restricted documents	Knowledge assistant	Medium	High	High	Data owner	Low	Quarterly	Permission-aware retrieval, classification
R-03	Model spend exceeds budget during peak	Model gateway	High	Medium	High	Platform owner	Medium	Monthly	Quotas, routing, budget alerts
R-04	Model provider deprecates the current model	AI orchestration layer	Medium	Medium	Medium	Engineering lead	Low	Quarterly	Gateway abstraction, eval suite for switch
R-05	Undocumented legacy rule lost in migration	Approval workflow	Medium	High	High	Business owner	Low	At each release	Characterization tests, parallel run

Appendix C

Sample architecture decision record

ADR-007: Route all model calls through a central gateway

Status: Accepted

Date: YYYY-MM-DD

01 **Context**

Model calls are made directly from four services with a shared API key.
We cannot enforce quotas, log usage, or switch providers.

02 **Decision**

All model calls will go through the model gateway service.

03 **Alternatives considered**

Keep direct calls and add a client library (rejected: no central enforcement)
Use the provider's own console limits (rejected: provider lock-in)

04 **Consequences**

One extra network hop per call; single place for limits, logs, and routing.

05 **Security implications**

Provider keys held only by the gateway; services use scoped internal tokens.

06 **Operational implications**

Gateway becomes critical path; requires its own SLO and on-call.

07 **Approval**

Solution architect, security lead, engineering lead.

Appendix D

Sample AI evaluation scorecard

Record one scorecard per release candidate, and store it as audit evidence:

	Evaluation criteria	Result / evidence record
01	Use case: the task the AI performs, and for whom.	
02	Model version and prompt version under test.	
03	Test dataset: name, version, size, and how it was built.	
04	Accuracy and groundedness: results against the agreed threshold.	
05	Safety findings: injection, jailbreak, and harmful-output test results.	
06	Bias or fairness findings, where the use case affects people differently.	
07	Cost per task at expected volume.	
08	Approval status: approved, approved with conditions, or rejected, with the approver's name.	

Appendix E

Glossary

AI agent:	software that uses a model to plan and take actions through tools, not just generate text.
CI/CD:	continuous integration and continuous delivery; automated build, test, and release pipelines.
Error budget:	the amount of unreliability an SLO allows over a period, used to balance features and stability.
Guardrails:	controls that constrain AI inputs, outputs, and actions.
Hallucination:	a model output that is fluent but unsupported or false.
Model gateway:	a service that brokers all model calls and enforces routing, limits, and logging.
RAG (retrieval-augmented generation):	supplying a model with retrieved documents so answers are grounded in approved sources.
SLI / SLO / SLA:	a service-level indicator (what you measure), objective (the target), and agreement (the contractual commitment).
Software bill of materials (SBOM):	an inventory of the components and dependencies in a piece of software.
Threat modeling:	a structured way to identify what could go wrong, who could cause it, and how to prevent it.
Vibe coding:	building software mainly through conversational prompts to AI tools, steering by outcome rather than line-by-line review.

References · 1 / 2

References

The standards and research below underpin the frameworks, controls, and recommendations in this paper.

AI governance and risk

ISO/IEC 42001:
the international management-system standard for AI governance, accountability, risk management, lifecycle controls, and continuous improvement.

NIST AI Risk Management Framework (AI RMF 1.0):
organizes AI risk work into four functions (Govern, Map, Measure, and Manage) for trustworthy AI systems.

OWASP Top 10 for LLM Applications:
catalogs the most critical security risks in generative-AI applications, from prompt injection to unbounded consumption.

Security governance and architecture

NIST Cybersecurity Framework 2.0:
structures enterprise cybersecurity around governing, identifying, protecting, detecting, responding, and recovering.

NIST SP 800-207, Zero Trust Architecture:
defines identity-first, least-privilege access that underpins API security and scoped AI-agent permissions.

CISA Secure by Design:
makes the case that secure defaults and customer protection must be built into software from the start, not added later.

OWASP Application Security Verification Standard (ASVS):
a practical checklist for specifying and verifying application-security controls across design, development, and testing.

CNCF Cloud Native Security Whitepaper:
guidance for securing cloud-native workloads across build, deployment, runtime, policy, and observability layers.

References · 2 / 2

References

Secure development and software supply chain

NIST Secure Software Development Framework, SP 800-218:
core secure-development practices that integrate into any SDLC to reduce vulnerabilities and improve response.

SLSA (Supply-chain Levels for Software Artifacts):
strengthens supply-chain security through build integrity, provenance, dependency verification, and protected CI/CD releases.

OpenSSF Scorecard:
automatically scores the security health of open-source dependencies to surface supply-chain risk.

Operations, reliability, and cost

NIST SP 800-61 Revision 3:
incident-response guidance for detection, containment, recovery, and post-incident improvement, aligned to NIST CSF 2.0.

Google SRE: Service Level Objectives:
explains how SLOs and error budgets balance reliability work against feature delivery.

FinOps Framework:
governs cloud and AI spend through usage visibility, cost allocation, budget ownership, and cost-per-outcome measurement.

Delivery research

DORA State of AI-assisted Software Development 2025:
finds that AI acts as an amplifier, magnifying the strengths and weaknesses of existing engineering systems.

Google Cloud: Announcing the 2025 DORA Report:
summarizes the report's headline findings on AI adoption, trust, and delivery performance (September 2025).

ABOUT TECHAHEAD

From AI pilots to production—securely, measurably, at scale.

TechAhead is an enterprise AI and product engineering partner that builds AI-native apps, platforms, and agentic systems. It helps organizations operationalize AI at scale, with the security, governance, and measurable business impact this paper describes, and moves enterprises from AI pilots to production through strategy, product engineering, cloud modernization, and managed delivery teams.

USA (HQ)
28720 Roadside Dr, STE 254,
Agoura Hills, CA 91301
+1 (818) 318-0727

India
6th Floor, Stellar 1423, Plot No. 3,
Sector 142, Noida, Delhi-NCR 201304
+91 120 6039900

UAE
API Trio Tower, Office 1105,
Sheikh Zayed Road, Al Barsha, Dubai
+971 58 592 9127

Sales@techaheadcorp.com

www.techaheadcorp.com