# Enterprise AI Developer Productivity Platform

## Product Requirements Document (PRD)

**(Note: Proprietary details have been modified for confidentiality)**

**Version:** 4.0
**Product Manager:** Ben Sweet
**Date:** (Revised) July 2025

## 1. Introduction

This PRD defines the requirements for a two-phase AI platform designed to accelerate software development across cloud-native enterprise environments. The system focuses on reducing cognitive load, surfacing hidden dependencies, and minimizing multi-system context switching. It is built to be both **practical at MVP scale** and **strategically extensible** as the organization matures in data readiness and operational sophistication.

This revision incorporates learnings from modern AI platform rollouts and explicitly addresses concerns that technical leadership often raises around over-engineering, observability gaps, premature fine-tuning, and unclear maintenance burden.

## 2. Product Vision

Developers today operate in fragmented ecosystems — dozens of repos, scattered documentation, Slack threads, and frequently fragile local environments. The vision of this platform is to become a **single, intelligent layer** that understands the organization's code, infrastructure, decisions, and historical context, enabling developers to work faster and with significantly fewer interruptions.

The long-term vision is not just "a smart assistant," but the **organization's reference AI platform** for knowledge retrieval, reasoning, automation, and structured agentic workflows.

## 3. Problems to Solve

The following issues routinely degrade developer velocity and morale:

1. **Context fragmentation:** Information is spread across GitHub, Slack, tickets, docs, and architecture decisions with no unified retrieval layer.
2. **Local environment fragility:** Minor misconfigurations often require hours of investigation, especially for new hires.
3. **Complex service dependencies:** Microservices introduce hidden cross-team dependencies that are difficult to reason about manually.
4. **Loss of institutional memory:** Architectural decisions, tradeoffs, and incident histories are rarely documented cohesively.
5. **Generic copilots lack organizational grounding:** Off-the-shelf copilots do not understand internal tools, conventions, or the architecture graph.

Solving these creates a measurable improvement in velocity and knowledge retention.

## 4. User Personas

### Primary: Software Developers (Backend, Frontend, Fullstack)

They need a frictionless, low-latency assistant that provides accurate answers directly inside existing workflows (IDE, Slack).

### Secondary: SRE / Platform / DevOps Engineers

They need faster diagnosis and structured guidance for environment issues, deployments, and cross-service operational problems.

### Tertiary: Engineering Managers & Architects

They require insight into system evolution, dependency risk, and the quality of decisions made across the org.

## 5. Key Goals & Success Metrics

### MVP Success Criteria

The MVP focuses on immediate, broad usefulness without requiring architectural overhaul. The goals below reflect pragmatic, measurable improvements:

- **25–40% reduction** in time spent searching for code, docs, Slack conversations, or architectural intent.
- **20–30% reduction** in onboarding time for new developers.
- At least **70% of developers** rate the MVP as "useful" or "very useful."
- **p95 latency < 4 seconds** for retrieval + synthesis queries, reinforcing trust in responsiveness.
- Infra cost remains within **$50–75 per developer/month**, consistent with modern RAG-first SaaS spend.

### Full-Scale Success Criteria

The full platform aims to influence architecture-level decision making and operational reliability:

- **50% reduction** in integration failures caused by unrecognized cross-service impacts.
- **60% utilization** of environment diagnostics and repair agents.
- **4× increase** in discoverability of architectural knowledge (decisions, ownership, dependencies).
- Demonstrated stability of autonomous workflows (CI/CD impact analyzer, meeting → ticket pipelines).

## 6. Scope Overview

The platform evolves in controlled phases to avoid over-engineering.

### Phase 1 — MVP

A focused, low-dependency product that brings immediate developer value with minimal operational complexity. It deliberately avoids distributed systems components (Kafka, Temporal, Neo4j) until proven necessary.

### Phase 2 — Full-Scale Release

An enterprise-grade architecture that introduces a knowledge graph, durable workflows, fine-tuned models for narrow tasks, and event-driven intelligence. This phase only unlocks once the org has validated real demand, collected sufficient data, and established operational foundations.

# 7. Functional Requirements

## Phase 1 — MVP Functional Requirements

The MVP is intentionally scoped to deliver high developer value with minimal operational complexity. It emphasizes retrieval, synthesis, developer interfaces, and basic diagnostics without introducing distributed systems or agentic automation.

### 1. Context-Aware Search & Synthesis

The MVP must unify context scattered across multiple tools and formats. The goal is not perfect omniscience, but **fast, explainable, trustworthy retrieval** that developers can verify via source attribution.

**MVP-R1.** The system will implement a RAG pipeline over source repositories, documentation repositories, Jira tickets, and Slack threads.

**MVP-R2.** The system will use AST-aware chunking for code in order to retrieve semantically meaningful units rather than arbitrary lines or token spans.

**MVP-R3.** The retrieval subsystem will support hybrid search using BM25 keyword retrieval and dense vector similarity search.

**MVP-R4.** The system will fuse retrieval results using Reciprocal Rank Fusion (RRF) to improve accuracy.

**MVP-R5.** The system will incorporate a hosted reranker model (e.g., Cohere Reranker 3) to refine final ranking before sending context to an LLM.

**MVP-R6.** The system will use hosted frontier models (e.g., Claude 3.5 Sonnet, GPT-4.1) for synthesis and reasoning.

**MVP-R7.** The system will perform strict data minimization, sending only the minimum relevant snippets to the LLM gateway to maintain compliance and protect proprietary information.

### 2. Developer UI (IDE + Slack)

User experience is the primary determinant of adoption. The system must integrate seamlessly into developer workflows and avoid forcing new behaviors.

**MVP-R8.** The platform will provide an IDE extension for VS Code and JetBrains that supports inline explanations and an expandable context panel.

**MVP-R9.** The IDE interface will allow developers to view retrieval sources for any generated response.

**MVP-R10.** The system will provide a Slack bot capable of answering context queries via slash commands.

**MVP-R11.** The Slack bot will generate threaded replies to maintain conversational organization and prevent channel disruption.

**MVP-R12.** The UI will avoid intrusive interactions such as modal popups or forced context windows.

## 3. Basic Environment Diagnostics

The MVP supports **diagnosis**, not **automated correction**, offering actionable next steps to shorten troubleshooting time.

**MVP-R13.** The system will run static environment checks for missing environment variables, port conflicts, dependency mismatches, and Docker Compose issues.

**MVP-R14.** Diagnostic results will be passed to an LLM for summarization into actionable suggestions.

**MVP-R15.** The system will NOT execute commands or perform modifications to the user's environment in MVP.

**MVP-R16.** The platform will allow developers to request "clarify this error" or "suggest next steps" from within the IDE.

## 4. LLM Gateway

The MVP must include a simple but reliable gateway that controls cost, routing, and provider failover.

**MVP-R17.** The system will route simple tasks to lower-cost models and complex reasoning tasks to higher-performance models.

**MVP-R18.** The gateway will enforce strict timeout limits for LLM calls.

**MVP-R19.** The gateway will enforce token budgeting on a per-request and per-team basis.

**MVP-R20.** The system will support automated provider failover when one LLM endpoint degrades.

**MVP-R21.** The gateway will disable vendor-side data retention where applicable.

## 5. Observability & Evaluation

Even the MVP must have a strong evaluation foundation to support trust and future expansion.

**MVP-R22.** The system will maintain a golden evaluation set for RAG accuracy and environment diagnostics accuracy.

**MVP-R23.** The system will expose dashboards showing query volume, latency, token usage, cache hit rate, and cost per team.

**MVP-R24.** The platform will support explicit feedback mechanisms (e.g., thumbs-up/down with optional notes).

## 6. Security

Security and privacy must be preserved through automatic inheritance of existing permissions.

**MVP-R25.** The system will enforce RBAC via inherited permissions from GitHub, Jira, and Slack.

**MVP-R26.** The system will perform sensitive data filtering or redaction for PII found in Slack or Jira content.

**MVP-R27.** The system will ensure that only sanitized context snippets are sent to external LLMs.

## 7. Out-of-Scope Items (By Intentional MVP Design)

The following items MUST NOT be included in the MVP to constrain complexity and allow 1–2 FTE operations.

**MVP-R28.** The MVP will NOT include Neo4j or any knowledge graph system.

**MVP-R29.** The MVP will NOT include Kafka or any event-streaming infrastructure.

**MVP-R30.** The MVP will NOT include Temporal or any durable workflow orchestrator.

**MVP-R31.** The MVP will NOT include agentic workflows or automated code/infra execution.

**MVP-R32.** The MVP will NOT include CI/CD pipeline integration.

**MVP-R33.** The MVP will NOT include fine-tuned models or self-hosted Llama instances.

**MVP-R34.** The MVP will NOT include automated environment remediation.

# Phase 2 — Full-Scale Functional Requirements

This phase expands the system into a full enterprise AI platform with graph reasoning, event-driven intelligence, durable automation, and optional fine-tuned specialist models.

## 1. Neo4j Enterprise Knowledge Graph

The knowledge graph becomes the system's structural backbone, enabling deep reasoning about dependencies, ownership, and architectural history.

**FS-R1.** The system will ingest data from static code analysis, service definitions, API schemas, and repository metadata into a Neo4j knowledge graph.

**FS-R2.** The system will ingest runtime signals such as OpenTelemetry traces to capture real inter-service call relationships.

**FS-R3.** The system will store architectural decisions, incident lineage, and ownership mappings as graph nodes and edges.

**FS-R4.** The knowledge graph will support graph queries that augment RAG retrieval and agent workflows.

**FS-R5.** The system will support dependency and risk analysis through KG queries.

## 2. Kafka Event Bus

Event-driven intelligence requires a unified stream of operational signals.

**FS-R6.** The system will integrate with Kafka to stream build events, PR merges, deployments, incidents, and environment errors.

**FS-R7.** The event bus will feed into automated workflows that detect breakages or changed dependencies.

**FS-R8.** The system will update the knowledge graph in near real-time based on incoming events.

## 3. Temporal Workflows

Durable workflows enable reliable agent execution with history, retries, and human approval gates.

**FS-R9.** The system will use Temporal to orchestrate multi-step environment repair workflows.

**FS-R10.** Temporal workflows will maintain full execution history for auditability.

**FS-R11.** Workflows will support human approval steps before applying changes to developer environments or repository state.

**FS-R12.** The system will use Temporal workflows to drive CI/CD impact analysis, knowledge synthesis, and documentation automation.

## 4. LangGraph Agents

Agents augment developer workflow by performing multi-step reasoning and controlled tool use.

**FS-R13.** The Environment Remediation Agent will diagnose environment problems, propose fixes, and validate them inside a sandbox.

**FS-R14.** The Environment Remediation Agent will request human approval prior to final execution.

**FS-R15.** The system will provide a Cross-Service Impact Analyzer Agent capable of identifying upstream and downstream service dependencies.

**FS-R16.** The Impact Analyzer Agent will use RAG, KG queries, and trace analysis to determine likely breakages.

**FS-R17.** The system will provide a Knowledge Synthesis Agent that converts meeting transcripts into structured decisions and links them to the KG.

## 5. Optional Fine-Tuned Llama Specialists

Fine-tuning is introduced only when justified by real usage patterns and dataset maturity.

**FS-R18.** Fine-tuning will require tasks with more than 10,000 daily invocations.

**FS-R19.** Fine-tuning will proceed only when labeled datasets meet inter-annotator agreement ≥ 0.7.

**FS-R20.** Real-time tasks will use Llama 3.x 8B models; large models (70B) will be reserved for batch operations.

**FS-R21.** Fine-tuned models will demonstrate ≥ 10% accuracy improvement over frontier LLMs via golden evaluation sets.

**FS-R22.** The platform will fall back to frontier models if fine-tuned specialists regress or fail evaluations.

## 6. Evaluation, Guardrails & Observability

Advanced evaluation and safety mechanisms ensure reliability and mitigate risk.

**FS-R23.** The system will maintain golden datasets for each capability (environment remediation, code review, impact analysis).

**FS-R24.** Guardrails will validate generated commands, code safety, schema adherence, and potential secret leakage.

**FS-R25.** The system will track online metrics such as acceptance rate, edit distance, error drift, and failure modes.

**FS-R26.** The evaluation pipeline will support automated regression detection on prompt or model changes.

# 8. Non-Functional Requirements

Expanded to reflect real-world constraints:

### Scalability

- MVP: Handles up to 10K daily queries using hosted LLMs and light infrastructure.
- Full Scale: Must support 100K+ queries with event-driven processing and caching.

### Latency

- MVP: p95 < 4s; acceptable for exploratory context queries.

- Full Scale: p95 < 2.5s through aggressive caching and local reranking.

### Reliability

- LLM Gateway uptime ≥ 99.5%.
- Temporal workflows must survive failures without human intervention.

### Security

- All inference pathways must respect privacy classifications.
- Air-gapped Llama mode available for highly regulated teams.

---

# 9. Architecture Summary

### MVP Architecture

Designed to be simple, maintainable, and cost-controlled:

- Weaviate vector DB
- Redis cache
- Backend API
- LLM gateway
- Slack bot
- IDE plugins
- Static environment checker
- Observatory dashboards + test suite

### Full-Scale Architecture

Adds components justified by validated demand:

- Neo4j knowledge graph
- Kafka event bus
- Temporal workflow engine
- LangGraph agent framework
- CI/CD integration
- Optional self-hosted Llama cluster
- Model lifecycle service

---

# 10. Risks & Mitigations

Expanded with rationale:

### Risk: Overbuilding early

*Many AI platforms fail because teams try to ship a fully agentic system at v1.*
Mitigation: Strict MVP scope; complex components introduced only when validated.

### Risk: Fine-tuning prematurely

Fine-tuning without clean data produces unreliable models.
Mitigation: Clear gating criteria and dataset quality standards.

### Risk: Developer adoption

Tools fail when they require behavior changes.
Mitigation: Deliver value directly in IDE + Slack; avoid portals; collect feedback loops.

### Risk: Cost overruns

Model usage can balloon unexpectedly.
Mitigation: Token budgeting, routing intelligence, caching targets, usage dashboards.

### Risk: Maintenance burden

Distributed systems can overwhelm small teams.
Mitigation: MVP requires only 1–2 FTE; full architecture modularized across 3–4 FTE.

---

# 11. Release Plan

Now with added narrative context:

### Phase 1 — MVP (12 weeks)

- Focused on search, synthesis, environment diagnostics, and strong UX integration.
- Pilot with 2–3 dev teams to collect usage data and identify the highest-value workflows.
- Establish baseline metrics for retrieval quality, time saved, and developer satisfaction.

### Phase 2 — Full Scale (6–12 months)

- Introduce KG, events, workflows, agents, and (optionally) fine-tuning.
- Expand coverage from a few teams to platform-wide adoption.
- Establish governance frameworks for agentic behavior and infra access.