AN APPROACH TO AI HIGH-VELOCITY DEVELOPMENT THROUGH SYSTEMATIC CONTEXT ENGINEERING: A CASE STUDY

Author:

Francesco Bisardi Independent Researcher

Francescobisardi@gmail.com

ABSTRACT

AI-assisted development promises substantial velocity gains, yet teams routinely fall short due to context loss and fragmented workflows. This paper introduces a two-layer context engineering architecture that treats context as a first-class system: (1) a declarative rule layer that encodes stable invariants, and (2) a programmatic Model Context Protocol (MCP) layer that exposes live project structure to AI agents. This architecture sits within a broader four-pillar synthesis consisting of an AI-optimized zero-friction stack, disciplined prompt workflows, the two-layer context system, and agentic orchestration patterns. We evaluate the approach through the development of a production-grade, multi-tenant SaaS platform (220k+ LoC) built by two part-time developers in fifteen weeks. Analysis of 3,676 AI-assisted sessions shows that Context Reuse Efficiency rose from 90.1% to 92.2% and Generative Amplification declined from 5.4% to 4.6%, consistent with a shift from generative substitution to retrieval-augmented reuse as context became systematized. Using a Design Science Research framework, we show that the two-layer pattern is feasible at production scale and produces measurable effects on development velocity. While based on a single case, the architecture and metrics establish a foundation for practitioners and researchers to validate and extend these practices across diverse contexts.

1. Introduction

An emerging shift in software development is underway. The emergence of "Vibe Coding", using AI agents to scaffold and write software has the potential to reduce development cycles from months to weeks. Nevertheless, a substantial execution gap remains between potential and practical outcomes. While industry reports often suggest consistent productivity gains, rigorous studies (Becker et al., 2025) show that unprepared teams often face slowdowns due to context loss, tooling friction, and review overhead.

The Context Engineering Challenge

Current research indicates that the effective utilization of large language models is fundamentally reliant on context management (Mei et al., 2025)(Du et al., 2025). Nonetheless, the software engineering community lacks thorough,

empirically-based recommendations on systematically designing context in AI-assisted production development. Although individual methods such as prompt engineering techniques, MCP servers, and agentic frameworks exist, they are fragmented and lack integration. There are two key gaps:

- **Execution and Integration Discrepancy:** Current activities are often depicted in isolation, leaving their systematic interrelations ambiguous.
- **Validation gap:** Predominantly anecdotal information exists; thorough rigorous case studies using these approaches are scarce.

Contribution

This work presents three key contributions to mitigate these deficiencies:

(1). Methodological Synthesis: We develop a practice-informed synthesis of context engineering principles within the proposed four-pillar approach, building on established work from rapid engineering, software architecture, and agentic AI systems. Our contribution is in:

- Systematizing how these strands are integrated across the proposed pillars
- Specifying concrete and interoperable technology choices that measurably reduce friction in AI-assisted workflows
- Formalizing a set of operational disciplines, distilled from our own applied experience, that prior work treats only implicitly or anecdotally

(2). Novel Two-Layer Context Architecture: We present and define a two-layer architecture for context engineering that integrates:

- **Declarative layer:** Encoding static rules for project-specific restrictions
- **Programmatic layer:** A bespoke MCP server that presents a live project structure (database schemas, UI components, migrations) for dynamic access.

(3) Empirical case study and metric operationalization. We instantiate this architecture in a production-grade, multi-tenant SaaS artifact (Bizie.app) and evaluate it through a naturalistic single-case study. We operationalize and report metrics such as Context Reuse Efficiency and Generative Amplification over 3,676 AI-assisted development sessions, providing empirical evidence on feasibility, velocity patterns, and the practical behavior of context-engineered workflows at scale.

Methodological Positioning: This study applies Design Science Research (DSR) methodology (Hevner et al., 2004; Peffers et al., 2007) to tackle a practical issue (context degradation in AI-assisted software development), through artifact

creation and naturalistic evaluation. In Design Science Research, contributions are determined by the demonstrated value of an artifact in usage, with more generalizability arising via community acceptance rather than initial controlled studies. Consistent with DSR guidance (Wieringa, 2014), our work is organized around three interrelated elements that correspond to the contributions above:

- A **design artifact**: the two-layer context architecture pattern, combining declarative rule sets with a programmatic Model Context Protocol (MCP) layer, embedded within the four-pillar framework;
- **An instantiation**: an implementation of this pattern in a Cursor IDE + bespoke MCP server environment, realized in the Bizie.app multi-tenant SaaS platform (~220k LoC);
- **Design knowledge:** practice-informed principles and operational disciplines for systematic context engineering, supported by metric operationalization (e.g., Context Reuse Efficiency, Generative Amplification) and empirical observations from 3,676 AI-assisted development sessions.

We frame this as exploratory design science research that advances practice-informed design knowledge rather than theory-testing experimental results. The architecture and its instantiation are evaluated through a naturalistic single-case study in an event-tech-productivity SaaS (TypeScript) context, structured around three research questions on production-scale feasibility (RQ1), observable velocity patterns (RQ2), and metric operationalization (RQ3). Comprehensive documentation is provided to facilitate replication and adaptation in alternative settings.

Case Study Context

Our working hypothesis is that AI-assisted development effectiveness depends primarily on project-specific context quality rather than model capabilities alone. This directly addresses the "Context Window Paradox," where larger context windows often degrade rather than improve performance (Mei et al., 2025; Du et al., 2025). We propose and evaluate a four-pillar approach to context management.

We examine this approach through Bizie.app, a research artifact, multi-tenant SaaS application for event-productivity tooling space (web and native mobile). Developed over 15 weeks by a two-person part-time team (a full-stack developer and a "vibe-coder") , the system comprises ~220k LoC implementing 78 features across CRM, event management, analytics, and AI workflows. By our assessment, this resulted in a feature set that overlaps substantially with established players in the event-productivity tooling space, who have spent years and millions in venture funding to reach a similar level of product maturity. While the implemented feature set addresses use cases comparable to existing event-productivity platforms, we emphasize that this single-case study demonstrates feasibility in one specific context and cannot establish broader generalizability.

Our case study demonstrates the feasibility of this approach in one context and

suggests directions for further validation and studies across diverse teams and domains. The proposed four-pillar approach entails:

- **A low-overhead, AI-optimized development environment:** A curated selection of tools that reduces infrastructure overhead and maximizes iteration speed.
- **A High-Discipline Operational Workflow:** A set of rigorous, repeatable practices for directing AI agents with precision, guiding quality is built-in, not bolted on.
- **Systematic Context Engineering:** The core contribution: a two-layer system of declarative rules and programmatic MCP server that solves "context rot" and provides the AI with comprehensive, current access to project information.
- **Agentic Orchestration.** Work is decomposed into role-based agents with explicit handoffs for analysis, implementation, and validation; long chains run in queues; completion is bound to verification artifacts.

The paper offers a detailed account, which may serve as a reference point for practitioners exploring AI-assisted development, while highlighting open questions for researchers.

2. Background and Related Work

Software development practices are shifting from long release cycles toward continuous iteration. At the center of this emerging model, there is "Vibe Coding".

2.1. The Vibe Coding Methodology The term "Vibe Coding," was introduced by Andrej Karpathy (Co-founder of OpenAI) in February 2025. This is an AI-dependent technique where the developer's role elevates from line-by-line implementation to systems architecture and AI direction. The developer provides a "High Quality Prompt" (encapsulating technical constraints, user stories, and business goals) and iterates with a code-generating LLM. SWE-bench Verified solved rate increased from 4.4 % to 71.7 % between 2023–2025 (*The 2025 AI Index Report | Stanford HAI*, 2025), (as of October 2025, Claude 4.5 reached 77.2%). Additionally, Dario Amodei in Sept, 2025 stated "70, 80, 90% of the code written in Anthropic is written by Claude" (Axios, 2025). Industry-wide adoption metrics have confirmed this shift: the 2025 DORA Report reveals 90% of software development professionals now integrate AI into core workflows, with 65% reporting heavy reliance and 80% experiencing enhanced productivity (Salva, 2025).

2.2. The growth of Vibe Coding Over the past year, there has been a rapid rise of platforms developed around this concept. This trend was catalyzed by

Devin AI in March 2024, marketed as the first autonomous AI software engineer, which demonstrated the potential for an AI to handle entire development tasks from a single prompt. Following this development, two distinct market categories emerged: prompt-to-app platforms and agentic environments, both showing growing commercial adoption.

Prompt-to-App Platforms (Rapid MVPs) are designed for non-technical founders, product managers, and rapid prototyping. Notable examples include Base44 and Lovable. **Agentic Environments** have emerged for professional developers, offering a suite of new agentic AI tools to augment and automate professional software development. Leading tools include Cursor, an AI-native IDE built from the ground up for agentic workflows; Claude Code by Anthropic,; and Replit AI Agent, a new entry that started as a Rapid MVP builder and is moving toward professional development.

Market maturity was underscored on September 23, 2025 when Cloudflare released vibeSDK, an open-source repository that allows anyone to deploy their own vibe-coding platform in a single click (https://github.com/cloudflare/vibesdk). This development effectively reduces barriers to entry for AI-assisted development platforms, and it suggests that the AI-native software development is not only an accelerating trend but rather a structural shift in the architecture of internet-based development infrastructure.

2.3. The Execution Gap Despite these advances, a significant disparity persists between the theoretical promise and practical outcomes achieved by most teams.

- **Industry reports** suggest that teams have achieved up to a **50% reduction in development time** (*Visma Develops New Code up to 50 Percent Faster With GitHub Copilot and Azure DevOps | Microsoft Customer Stories*, 2025)
- Academic studies indicate that without the right processes, using AI tooling can **increase task completion time by 19% (so it's slowing down the process)** due to the overhead of correcting AI errors, caused by a lack of context and PR reviews (Becker et al., 2025). This productivity paradox extends beyond completion time: evaluation of LLM agents on repository-level tasks demonstrates that sophisticated prompt engineering techniques, which improve performance by 19% on simple benchmarks, yield only 1.6% improvements at repository scale, revealing fundamental limitations in current approaches to code generation at scale (SECRePoBench: Benchmarking LLMs for Secure Code Generation in Real-World Repositories, 2025)

To narrow this gap, teams may benefit from moving beyond ad-hoc prompting toward more structured approaches. Our work examines one concrete approach to organizing development in this emerging AI-native landscape. As tools and

foundational technologies continue to evolve, the demand for comprehensive methodological frameworks and appropriately skilled technical staff becomes increasingly urgent. This transition implies not incremental, but systematic re-training of existing technical workforces.

2.4. The Context Window: An LLM's Aperture on Reality At its core, a Large Language Model operates within a finite **"context window"**, the total amount of information it can consider at any one time. This can be thought of as the model's working memory, its aperture on the reality of a given problem. Base model:

- GPT-5 Context windows: 272k tokens - (*Introducing GPT-5 for Developers*, 2025)
- Gemini 2.5 Pro: 1M tokens - (*Gemini Models*, 2025)
- Claude 4.5: 200k (context aware) - (*Claude Docs*, 2025)

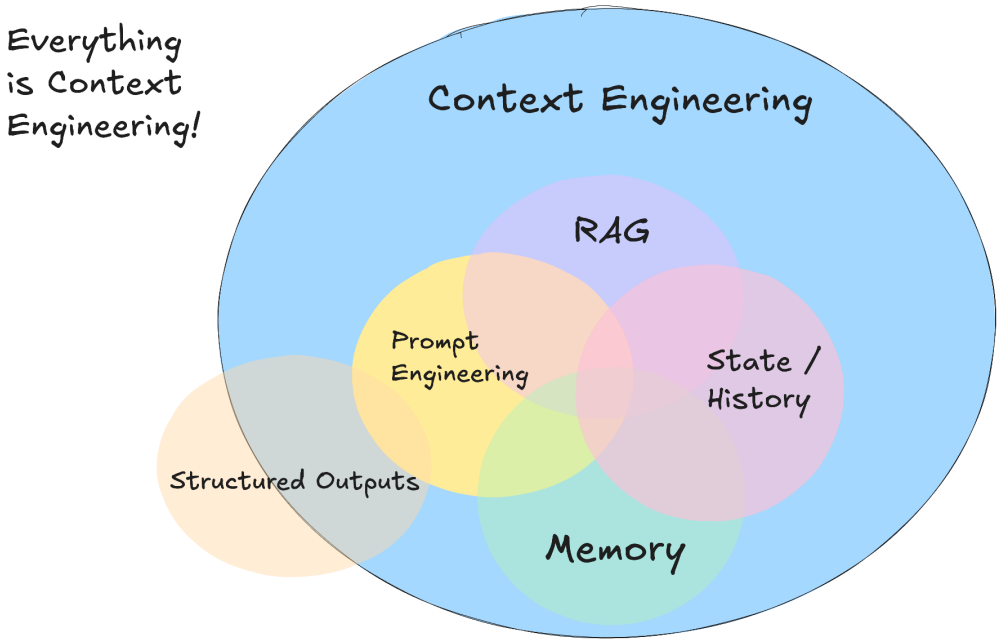
This expansion, however, has given rise to the "Context Window Paradox": a phenomenon well-documented in recent long-context studies (Bertsch et al., 2025; Baek et al., 2025), simply having a larger window does not guarantee better results (*LLM Context Management: How to Improve Performance and Lower Costs*, 2025). In fact, naively filling a large context window often leads to significant performance degradation (Chroma Research, 2025; Adaline Labs, 2025). Amazon Science research published in 2025 demonstrates that as context length increases, performance degrades substantially (13.9%–85%) even when models can perfectly retrieve all relevant information (Du et al., 2025). The Chroma study examining 18 different models revealed that model performance can drop from ~95% to ~60-70% on longer inputs containing semantically relevant content and distractors (Chroma Research, 2025). This is due to several well-understood failure modes:

- **Degraded Signal-to-Noise Ratio**: A software project's repository contains vastly more noise (boilerplate, unrelated modules, old code) than signal for any specific task (Heineman et al., 2025.; MarkTechPost, 2025). Heineman et al. in 2025 demonstrates that benchmarks with higher signal-to-noise ratio correlate strongly with better decision accuracy (R^2=0.626), while low-SNR contexts force models to waste computational resources on irrelevant information and produce unfocused, error-prone responses (Heineman et al., 2025).
- **Architectural Inconsistency & "Context Rot":** This is the most significant challenge in long-term software projects (Adaline Labs, 2025; Valyu Network, 2025). Context rot, formally defined as "the systematic degradation of model accuracy and reliability as input context length increases" (Valyu Network, 2025), occurs when long conversations drift off course because models cling

to outdated or irrelevant parts of the discussion. Adaline Labs' 2025 research shows that every extra page of context reduces accuracy beyond a certain point, with performance degradation manifesting as omissions, confusion, or topic drift in model outputs (Adaline Labs, 2025).

- **The "Lost-in-the-Middle" Phenomenon:** Academic research confirms that large language models tend to pay more attention to information at the very beginning and very end of long contexts, ignoring critical details buried in the middle (ACL Findings, 2025; University of Washington et al., 2024). Recent work published in ACL Findings 2025 introduces the related "Lost in the Distance" phenomenon, where LLMs struggle to retrieve relational knowledge when intermediate noise tokens interfere with proper attention between knowledge components (ACL Findings, 2025). This pattern holds even for explicitly long-context models, with research showing that LLMs are more likely to recall information positioned at the start or end of input, with a tendency to overlook content in the middle (AI Planet, 2023; The Decoder, 2023).
- **Factual Drift and Hallucination:** Without robust and immediate grounding in project-specific context, large language models "drift" towards generic patterns in their training data (Wei et al., 2025; Lakera AI, 2025). Research on "Contextual Perturbation and Representation Drift" shows that context perturbations trigger systematic drift in LLM hidden representations leading to hallucinations (Wei et al., 2025). The research indicates that the accumulation of irrelevant information leads large language models to become embedded in self-created erroneous contexts. This results in attention patterns stabilizing, causing generated hallucinations to solidify, thereby rendering traditional correction methods ineffective (Wei et al., 2025).

Therefore, high-velocity development is not a matter of simply using AI, but of mastering the discipline of engineering the context in which it operates (Anthropic, 2025; Valyu Network, 2025). The proposed approach in the paper was designed explicitly to solve these challenges through a two-layer approach with declarative context repo Native MCP.



Context Engineering. Humanlayer. (2025). *12-factor-agents/content/factor-03-own-your-context-window.md at main · humanlayer/12-factor-agents*. GitHub. https://github.com/humanlayer/12-factor-agents/blob/main/content/factor-03-own-your-context-window.md

2.5 What MCP is:

Since it was open-sourced by Anthropic in November 2024 with SDKs and reference servers, MCP is becoming the de facto way to connect LLMs to tools, data, and workflows without a thicket of bespoke adapters. Often referred to as the USB-C port for AI applications. In short, MCP is a client–server protocol: an LLM "host" (e.g., an app with an agent runtime) launches one client per server. MCP Servers run locally or remotely and self-describe a machine-readable catalog of capabilities, organized into three primitives: **(1)** tools (actions), **(2)** resources (data/context), and **(3)** prompts (templated workflows).

Feature	MCP	API
Purpose	*Standardizes* AI model integration with external systems.	Facilitates communication between different software applications.
Data Handling	Context-aware data exchange.	Data exchange based on predefined endpoints.
Flexibility	Designed for dynamic, context-sensitive interactions.	Structured and static interactions.
Security	A secure, two-way connection with contextual metadata.	Depends on implementation; it may require additional security measures.
Scalability	Supports complex, multi-source integrations.	Scales based on the number of endpoints and requests.

2.6 Design Science Research in Software Engineering

In software engineering, DSR has effectively progressed various domains through the documentation and implementation of patterns prior to formal validation, including architecture patterns (Shaw, 1996), development methodologies and tool design (Ko et al., 2018). Peffers et al. (2007) argue that contributions in Design Science Research (DSR) do not require immediate generalizability; rather, demonstrating feasibility and utility through implementation is a valid contribution, with broader validation arising from community adoption.

This study adheres to recognized DSR evaluation strategies (Venable et al., 2016), employing a naturalistic evaluation through an observational case study in a production setting. This approach is suitable for exploratory research aimed at refining the design requirements of the artifact through its usage. We acknowledge the conventional DSR tradeoff: a profound comprehension of a single context instead of superficial validation across multiple contexts. This positions our two-layer architecture as a design pattern for community assessment rather than an established best practice. Based on Wieringa (2014) we identified the following relevant aspects of our contribution:

- Design artifacts: the two-layer context architecture pattern, embedded within a four-pillar framework for AI-assisted development

- Instantiations: an implementation of this pattern in a Cursor + MCP–based development environment, concretized in the Bizie.app multi-tenant SaaS platform;
- Design knowledge: practice-informed principles and operational disciplines for systematic context engineering, which can be adapted to alternative stacks and organizational settings.

Our work is organized to delineate these concerns, allowing others to implement the architecture in alternative ways while maintaining fundamental design principles.

3. Framework Design and Implementation

Design Science Framing: this section outlines our main research contribution: the two-layer context architecture (Sections 3.3-3.4) and details its implementation within a four-pillar framework. In accordance with Design Science practices, we differentiate between the architectural pattern (generalizable design knowledge) and our specific implementation choices (a singular instantiation). The architecture responds to requirements that arise from context degradation issues (Section 2.4); its implementation illustrates feasibility within a specific production context; and Section 4 evaluates utility via naturalistic observation addressing three research questions: production-scale feasibility (RQ1), observable velocity patterns (RQ2), and metric operationalization (RQ3).

To validate production-scale applicability, we implemented a research artifact enterprise-grade SaaS platform (220k LoC, 78 features, 15 weeks, two-person part-time team). Our framework integrates four pillars drawing on established techniques:

- Pillar 1: A low-overhead, AI-optimized development environment Selection
- Pillar 2: synthesizes prompt engineering research (Chen et al., 2023) with IDE-specific practices
- Pillar 3: introduces our novel two-layer context architecture
- Pillar 4: integrates agentic patterns from spec-kit, Archon, and PRPs frameworks into Cursor

Our contribution in this section is threefold: (1) systematic integration of these components into a cohesive workflow, (2) formalization of the two-layer context pattern, and (3) detailed documentation of implementation decisions, trade-offs, and operational disciplines. We present this as "our approach" rather than prescriptive rules, as we acknowledge that alternative scenarios may be equally valid. The described approach was developed iteratively during the build of the research artifact case study (Section 4), and refined through 3,676 development sessions over 15 weeks.

3.1. Pillar 1: A low-overhead, AI-optimized Development Environment Selection **Design Principle:** We argue that infrastructure complexity accumulates over numerous AI-assisted iterations, resulting in inefficiencies that undermines productivity gains from AI. Our stack selection prioritized: (1) comprehensive documentation for LLM consumption, (2) minimal configuration overhead, and (3) rapid feedback cycles. **Note on Generality:** This specific stack worked well for our use case (web/mobile SaaS, TypeScript). Other contexts may benefit from different choices. The overarching principle is: choose tools that reduce general infrastructure work rather than focusing on specific tools.

Component	Technology	Velocity Mechanism
Build System	Turborepo Monorepo + Mobile Repo	Supports parallelized builds with shared caching, providing rapid feedback cycles during development.
Framework	Next.js + React + Expo	A modern full-stack framework featuring optimized bundling, facilitating rapid frontend and API development while Expo abstracts low-level mobile build processes.
Backend	Supabase (BaaS) + Vercel	In our experience, this combination provided a fully managed, end-to-end platform that minimized infrastructure overhead. Supabase provides the instant state layer (PostgreSQL, authentication, storage), eliminating months of backend setup. Vercel provides the serverless compute layer and automates the full delivery pipeline through Git-driven workflows and instant Preview Environments.
AI Orchestration	LangChain + LangGraph + n8n	LangChain for building LLM agents; LangGraph for graph-structured workflows; n8n for low-code AI automation.
Language	Full-stack TypeScript + Zod	Provides end-to-end type safety, catches errors at compile time, and enables AI agents to refactor code reliably.
UI	Shadcn UI + Tailwind CSS	Supports rapid, consistent UI development through utility-first styling and composable components.
Testing	Playwright (E2E)	Provides deterministic, parallelized cross-browser testing with CI-grade speed, further enhanced by MCP-based testing agents (Oct 2025).

Some of these components are well-known and leveraged by many Vibe coding platforms. Although implementation choices differ across platforms, this combination removes most general infrastructure work in software development allowing the team and their AI tools to focus entirely on feature delivery.

Expo (mobile development environment used by Bolt.new and Replit (*Expo for Mobile Apps - Bolt*, 2025), (*From Idea to App With Replit and Expo*, 2025) provides a native-first mobile workflow that offers broader device support than PWA-based approaches such as Lovable, Base44, or v0 by Vercel. ***Tailwind CSS and shadcn/ui***, used by Lovable and v0, provide fast, consistent UI development, while Supabase (used by Lovable (*Supabase Integration - Lovable Documentation*, 2025)) offers a reliable backend foundation. Together, these components create a modern, high-velocity stack optimized for both speed and user experience.

We note that pre-configured boilerplates (e.g., Vercel templates) may provide similar initial acceleration, representing an alternative starting point. (Find Your Template, 2025)

3.2. Pillar 2: Disciplined Prompt Engineering Workflow Design Principle: We synthesized prompt engineering research (Chen et al., 2023; Vilakati, 2025) into operational practices tailored for IDE-based development. While techniques like chain-of-thought and few-shot prompting are well-established, their application within long-running Cursor sessions required adaptation. The practices below represent our refinement of general principles into Cursor-specific workflows, informed by 3,676 development sessions.

A performant stack alone is insufficient without skilled prompt formulation; thus, the second pillar focuses on methodical prompt engineering. Effective Vibe Coding is not about vague requests; it depends on explicit, structured prompt design on the other hand, uncontrolled automation increases maintenance cost and erodes net productivity. Comprehensive reviews establish prompt engineering as the process of structuring inputs to maximize utility and accuracy, encompassing techniques such as self-consistency, chain-of-thought, and generated knowledge (Chen et al., 2023). Empirical evaluations demonstrate that hybrid prompting (combining explicit instructions, reasoning scaffolds, and format constraints) consistently produces more accurate and interpretable results than zero-shot, explicit instruction, or chain-of-thought approaches alone (Vilakati, 2025). This pillar formalizes prompt construction as a repeatable engineering discipline rather than an intuitive process. The approach requires embedding multifunctional context within each prompt by specifying:

- User Stories and Business Goals.
- Precise UI/UX Descriptions.
- Technical Constraints (referencing the specific stack in Pillar 1).

- Data models
- Testing Plans.

Cursor's best practices, Agent rules, MCPs

We operationalized these concepts through specific tool configurations and procedural disciplines in our Cursor-based workflow. Cursor, an AI-native IDE, served as the primary implementation environment. Although effective independently, Cursor's capabilities are maximized when embedded within a structured workflows. The subsequent practices detail the tactical implementation of our approach, aimed at maximizing signal-to-noise ratio and maintaining consistent high-quality AI outputs.

Operational Practices applied in this Study

From Research to Practice: The prompt engineering literature provides conceptual guidance (e.g., "provide examples," "structure reasoning"). Our contribution is translating these concepts into repeatable operational disciplines: *how* to structure Cursor sessions, *when* to use which model, *how* to integrate external context sources. These disciplines emerged iteratively, we do not claim they are the only valid approach.

The workflow is organized around four core disciplines: establishing rich contextual foundations, enforcing task-scoped workspaces, orchestrating tools with explicit intent, Accelerate the Debugging and Verification Loop.

1. Establish a Foundation of Semantic Context

We observed that AI output quality correlated with input context quality.

- **Activate Full-Repository Indexing:** The cornerstone of the workflow is enabling Cursor's codebase indexing. This creates a semantic understanding of the entire project, allowing the agent to perform intelligent retrieval.
- **Enhance Retrieval with Metadata:** To guide the retrieval process, annotate key files with simple, machine-readable tags. A comment like *// @module:authentication* provides a strong signal to the RAG system, ensuring it can precisely locate the most relevant context for a given task. Tagging can be set as a rule or Cursor command for the agent.
- **Integrate External Knowledge Sources:** LLMs knowledge can be outdated, using Cursor's *@docs* command, dynamically pull in indexed documentation for critical services giving the agent immediate, accurate context on the APIs it needs to interact with.

2. Enforce Disciplined Task-Scoped Workspaces

To prevent context bleed and maintain focus, each development task is treated as an atomic unit of work with its own dedicated agent.

- **Each task was isolated in a dedicated session to avoid context bleed:** Each feature or defect is addressed in a new conversational session, aligning with the isolation strategy to avoid cross-task context contamination.
- **Integrate Version Control for Atomic Commits:** Cursor's integrated restore features and Git are used to create atomic commits with clear descriptions. This allows us to create small, atomic, AI-assisted commits with clear descriptions, ensuring every change is traceable, understandable, and easily reversible.

3. Orchestrate Tools and Models with Intent

Effective AI-assisted development requires treating the AI less like an autocompleter and more like a specialist that must be directed.

- **Select the Right Model for the Job:** A multi-model strategy is used to match model capabilities to task complexity. For high-level tasks requiring deep reasoning, such as architectural planning or complex code analysis/bugs, reasoning models Claude 4.5, GPT codex or Gemini 2.5 Pro (1M token window) tend to excel. Non-reasoning models are used for small action-specific changes. While Cursor's Auto-select feature is a good compromise for low-medium-complexity tasks, Claude 4.5 often remains the best model.
- **Target UI Development with Precision:** When working on the frontend, we use Chrome / Cursor Browser MCP's "inspect" feature. This allows to ground the AI's context in a specific visual element on a rendered page, moving from abstract descriptions like "the login button" to a concrete, unambiguous target.
- **UI/UX Design partner:** The design process followed a component-first, system-driven approach inspired by *21st.dev* and other open source libraries, emphasizing clarity, hierarchy, and motion as functional cues rather than decoration. Each design decision must be directly mapped to reusable code components, allowing the AI to generate and iterate on interfaces with precision and visual consistency.
- **Activate Programmatic Context via MCPs:** Although the model can activate MCPs automatically, explicit invocation improves consistency and result quality.
- **Apply Weighted Governance to Agent Actions:** AI autonomy is governed through Cursor's task manager to balance speed and safety. High-risk actions (e.g., database migrations,

dependency installs) should be set as an "Ask Everytime" human checkpoint, while low-risk tasks (e.g., code refactoring) use an allowlist for faster automation. This risk-based control ensures oversight where needed and efficiency elsewhere.

- **Define and Access Cursor Commands. [.cursor/commands]** Cursor Commands are structured, predeclared task shortcuts that encapsulate complex developer intentions as reusable directives.
- **Use Cursor Command Queue for Session Autonomy** Long development sessions often exceed a single context window. Cursor's command queue feature allows the agent to stack and execute sequenced actions (e.g., "refactor → run tests →create docs →commit ") without continuous manual prompting. This preserves state continuity while reducing human-interruption frequency

4. Accelerate the Debugging and Verification Loop

Even with high-quality context, debugging is an inevitable part of development. The methodology transforms this phase from a manual bottleneck into a rapid, AI-assisted feedback cycle.

- **Use LLMs with Active Browsing for UI-Level Validation:** Incorporating an LLM with active browsing capability enables automated end-to-end verification of application interfaces. Rather than relying solely on static unit or API tests, the agent can interact directly with rendered web or mobile components (clicking buttons, validating DOM states, verifying visual changes, and capturing console outputs). This approach transforms traditional manual QA into a live, model-driven inspection layer, ensuring that generated code not only compiles but behaves correctly in its operational environment. When combined with frameworks such as Playwright MCP or Browser MCP, it allows the agent to reason over the full visual and interactive context of the application under test. Note: Cursor released a beta version of its own browsing function.
- **AI-Assisted Diagnostics:** When a function behaves unexpectedly, instead of manually reviewing the code with console.log statements, the user can instruct the agent to perform targeted diagnostics. A simple prompt like, *"This server action is failing silently. Add logging to the terminal to trace the user ID and their permissions as they enter the function,"* leverages the agent's understanding of the code to place the most effective diagnostic lines instantly.
- **Interactive Log Analysis:** This is the core of the accelerated

loop. Once an error is produced, the full, raw output from the terminal or browser developer console is fed directly back to the agent. Models like Claude 4.5 tend to be exceptionally skilled at this. By providing the error logs, stack trace, and relevant code in the same context, the Agent can often identify the root cause and propose a precise fix in seconds, closing the loop between execution, failure, and resolution.

Used MCPs

Chrome / Browser MCP	Chrome DevTools / Browser MCP gives AI coding assistants "eyes" in the browser by connecting them directly to the browser's debugging and automation capabilities. It acts as a bridge between AI agents.
Playwright MCP	Playwright MCP enables AI assistants to perform browser and API testing automation using Playwright's cross-browser capabilities. It provides structured accessibility-tree interactions instead of pixel-based approaches. It also includes three testing agents (Agents | Playwright, 2025).
Supabase MCP	Supabase MCP serves as a secure bridge connecting AI assistants to Supabase databases, enabling controlled querying, analysis, and database operations.
Stripe MCP	Stripe MCP allows AI agents to interact with Stripe's payment API through natural language commands, automating payment processing, customer management, and financial operations.
Native Repo MCP	Covered in detail in Pillar 3 (next chapter).

Strategic Consideration: Managing Tool Capacity. Many LLMs show performance degradation when exposed to an excessive number of tools, often beyond a threshold of 80 (Cursor, September 2025),(Code Execution With MCP: Building More Efficient AI Agents, 2025). Therefore, its standard practice is to enable only the MCPs essential for the current task. For most development workflows, this involves keeping the Native Repo MCP persistently active, while selectively enabling more comprehensive servers, like the Supabase, Stripe MCP, only when their full capabilities are required.

3.3. Pillar 3a: Two-Layer Context Architecture: Static Context

The third pillar addresses what we identified as a central challenge in our

development process: effectively managing the model's context window. While model capability often gets the spotlight, our experience building the research artifact reinforced our hypothesis that context quality significantly influences development outcomes, consistent with recent research on context engineering (Mei et al., 2025). The foundation of the proposed context strategy is a set of static, human-authored rules that define the project's non-negotiable standards. These rules act as guardrails, ensuring that every AI-generated output adheres to the core architectural principles, security policies, and coding conventions from the outset.

The research artifact project maintained a comprehensive library of rules within a .cursor/rules directory. This allowed the used primary AI-native IDE, Cursor, to inject these constraints into the context of relevant tasks automatically (if set as always on, or called upon *@ruleName)*.

Cursor Rule Type	**Description**
Always	Always included in the model context.
Auto	Included when files matching a glob pattern are referenced (e.g., *globs: ["**/*.ts", "**/*.tsx"]*).
Manual	Only included when explicitly referenced using `@ruleName`.

Example:

```
---
description: RPC Service boilerplate
globs: ["**/*.ts", "**/*.tsx"]
alwaysApply: false
---

- Use our internal RPC pattern when defining services
- Always use snake_case for service names.

@service-template.ts
```

This rulebook provides explicit guidance on critical domains, including:

- Specifies rules for interacting with the Supabase backend.
- Enforces security best practices, such as input validation.
- Defines standards for component structure.
- Outlines the repository's file organization to ensure consistency.

- Atomic Git, lint rules, avoid local DB reset, clean up of temporary files to keep RAG clean.

This declarative layer served as our primary quality control mechanism, encoding project-specific constraints in a machine-readable format.

3.4. Pillar 3b: Two-Layer Context Architecture: Dynamic Context While declarative rules are powerful, they are static. To provide the AI with a real-time, dynamic understanding of the project, a custom **MCP** server was implemented. As mentioned earlier, MCPs are organized into three primitives: **(1)** tools (actions), **(2)** resources (data/context), and **(3)** prompts (templated workflows)

This server provides an active, structured API for the AI agent to query, allowing it to pull in only the most relevant, up-to-date information for a given task. This approach maximizes the signal-to-noise ratio inside the context window.

The native repo MCP server grants AI agents programmatic access to:

- **Database Schema & Functions:** Real-time access to all Supabase schema files and Postgres functions.
- **UI Component Library:** A complete, searchable catalog of all available UI components.
- **Database Migrations & Development Scripts:** The ability to understand and utilize our project's specific workflows.

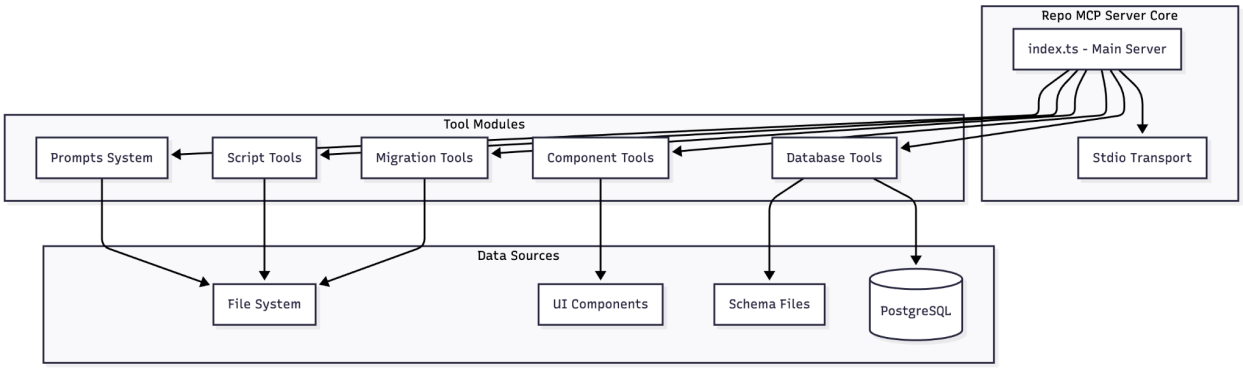
This programmatic layer enables the model to retrieve and reason over project data programmatically rather than through static inspection.

3.5. Activating Context with Specification Templates The final phase involves activating this rich, two-layer contextual framework through high-level, reusable specifications. The MCP Server operates as a bridge between AI assistants (e.g., Cursor, Claude Code) and a production codebase, exposing a catalog of structured capabilities organized into six modular tool categories: Database, Components, Migrations, Scripts, and Prompts. Each module provides machine-readable access to project metadata (database schemas, UI libraries, code-quality scripts) enabling real-time, context-aware model reasoning.

The architecture adheres to the canonical MCP design pattern, leveraging a central Stdio-based server core that communicates with AI clients through defined protocol schemas. The modular design allows individual tool modules to self-describe their capabilities and expose standardized actions, resources, and templated prompts. In practice, the server supports live database context, component dependency mapping, migration tracking, automated QA script discovery, and code security.

The MCP server hosts a collection of pre-constructed prompt templates, each systematically designed to address complex and recurrent development tasks.

MCP Primitives		
Tools (Actions)	**Resources (data/context)**	**Prompts (templated workflows)**
• DB actions (read to get live data) • Components • Scripts (code quality)	• Next.js routes • UI components • Design tokens • DB schemas, migrations, tables, types, functions *(ideal state)* • Security/policy rules	• DB work (schema, migration) • Code quality templates • Architecture guidance



3.6. The Pillar 4: The Agentic Orchestration Patterns While the three pillars provide a methodology for an individual or a small team to achieve high development velocity, the next frontier is scaling this capability to tackle enterprise-grade complexity with institutional rigor. This requires moving beyond a single developer-agent interaction to an **Agentic Orchestration Framework**: a system where a team of specialized AI agents collaborates on a project under the direction of a human operator and a master orchestrator.

This approach represents a more structured application of context-engineering principles, drawing on emerging agentic frameworks. It treats software development not as a series of discrete prompts, but as a managed, automated workflow executed by a digital software team. Across pioneering open-source frameworks like spec-kit (from GitHub), PRPs-agentic-eng, Archon, AGENTS.md and claude-task-master, a shared philosophy has emerged: prepare rich context upfront, plan thoroughly, enforce validation, and leverage specialized tools and agents. This approach is built on four core principles:

Design Principle 1: Decouple Planning from Execution
The single most common failure mode in agentic systems is premature execution

based on incomplete context. The forth pillar approach enforces a strict two-phase process, separating the *what* and *why* from the *how*.

- **Phase 1: Agentic Planning & Specification.** Before any implementation code is written, we created AGENTS.md ("README for AI agents") and deployed a team of "planning agents" is deployed to produce a central, machine-readable artifact. Frameworks like spec-kit achieve this with a /speckit.specify command, while others generate a **Product Requirement Prompt (PRP)** that bundles a PRD with curated codebase intelligence. This artifact is the result of deep analysis, external research, and blueprint creation, codifying intent and reducing downstream hallucination. Cursor's Plan Mode, released September 29, 2025, represents an early implementation of this phased approach.
- **Phase 2: Context-Engineered Development.** Only after the plan is validated does the Orchestrator activate the "development agents." The specification or PRP from the planning phase becomes the non-negotiable context for this phase, ensuring every agent works from the same source of truth and follows a pre-defined plan, often recorded in a TodoWrite list to track progress.

Design Principle 2: Standardize Context Management.
Pillar 4 does not introduce a new context mechanism; it reuses the two-layer architecture from Pillar 3 as the substrate for multi-agent workflows. At the static layer, non-negotiable principles are encoded as shared rulebooks (e.g., our .cursor/rules file), analogous to GitHub *spec-kit*, which operates at the "constitution" level via a *constitution.md* that specifies global project norms and patterns. At the dynamic layer, agents draw on live, task-specific knowledge sources (e.g., our native repo MCP), analogous to *Archon*, which provides a centralized "living context" hub backed by vector search for code, docs, and metadata. We reference spec-kit and Archon as alternative instantiations of the same static–dynamic pattern rather than components of our artifact. The contribution here is not a new context model, but the design principle and standardization of this static–dynamic context as shared infrastructure for an entire agent team instead of per-prompt configuration.

Design Principle 3: Orchestrate Hierarchical, Role-Based Agent Teams
Just as human teams have specialized roles, this framework utilizes a team of agents orchestrated through hierarchical delegation.

- **Specialized Sub-Agents:** The system uses distinct agents for specific tasks: a Codebase Analyst to understand existing code, a *Research Agent* for external documentation, and a *Validator Agent* for code review. This specialization encapsulates knowledge and improves focus. Maintaining velocity also requires

a *Maintenance or Cleanup Agent*: a specialized agent invoked after a feature's completion to review the resulting codebase and remove deprecated or redundant artifacts. This process reduces residual noise, minimizes obsolete dependencies, and keeps the contextual footprint of the project compact. The result is a leaner, more coherent repository that lowers the amount of context the model must load for subsequent tasks, thereby preserving focus and consistency across development cycles. Specialized agentic approaches designed for repository-scale understanding demonstrate the necessity of such orchestration: Kodezi Chronos, combining Adaptive Graph-Guided Retrieval with Persistent Debug Memory, achieves 67.3% fix accuracy on real-world debugging versus 14.2% for Claude and 13.8% for GPT-4.1 (Khan et al., 2025). Within Cursor, agents can be replicated through explicit rules and queued command invocations, with optional use of external tools executed directly inside the IDE.

- **The "Agent-as-a-Tool" Pattern:** A primary agent can delegate tasks to sub-agents. For example, a primary planning agent can invoke a specialized research agent to analyze a competitor's API, passing along the necessary context. This hierarchical model, managed by an Orchestrator, allows for the composition of complex workflows from simple, modular agent capabilities.

Design Principle 4: Embed Continuous Validation and Quality Gating To ensure that velocity does not compromise quality, the framework embeds automated validation at every stage, with a dedicated QA Agent acting as an impartial gatekeeper.

- **Embedded Validation Loops:** The core planning artifacts (PRPs or specifications) include a "validation loop" section. This explicitly lists the commands the agent *must* run to verify its work (such as syntax checkers (ESLint, Zod, Prettier), unit tests (Jest), and integration tests) before a task can be marked as complete.
- **Proactive Risk Management:** Before implementation, tasks are analyzed for risk. Frameworks like TaskMaster starts with an analyze-complexity set of workflow to estimate difficulty and recommend splitting complex tasks, proactively managing bottlenecks.
- **Self-Consistent Analysis:** Advanced frameworks provide tools for "meta-cognition". spec-kit's /speckit.analyze command checks for consistency across the specification, plan, and task artifacts, catching potential bugs.

Our synthesis of this approach is informed by the foundational work of several

pioneering open-source projects. For those seeking to go deeper, these repositories represent some of the current advances in agentic software engineering:

- **github/spec-kit:** A GitHub toolkit for spec-driven development that helps AI agents and humans build software from structured specifications.
- **eyaltoledano/claude-task-master:** Task orchestration system for Claude-based AI development that focuses on structuring and managing complex tasks (not specifically risk analysis).
- **Wirasm/PRPs-agentic-eng:** Framework using Product Requirement Prompts (PRPs) to guide agentic engineering workflows.
- **coleam00/Archon:** Centralized command-center framework for coordinating multi-agent systems, context, and knowledge management.
- **coleam00/context-engineering-intro:** Introductory guide to context engineering with examples, templates, and MCP server implementations.
- **openai/agents.md:** Simple, tool-agnostic format for AGENTS.md files that acts as a "README for agents," giving agents a predictable place for build, test, and workflow instructions.

In the beginning of October Cursor released *Plan Mode* which introduces the company's attempt to orchestrate a deliberate, two-phase workflow: plan first, act later. Before generating code, the agent conducts a structured analysis of the repository, retrieving and summarizing only the most relevant files, functions, and documentation. At that point, it produces a reviewable Markdown plan that describes each step, file, and dependency affected. This explicit planning layer transforms the AI from an auto-coder into a context-aware collaborator, while the Markdown plan still needs review, paired with native repo MCP for specific repo context, provides measurable improvements and comparable results to more cutting edge frameworks. It's worth mentioning that CLI like Claude code or OpenAI codex can also be run within Cursor.

3.7. The Hand-off: Full-Stack Developer to vibe-coder and vibe-coder to Full-Stack Developer

We used a two-function execution model to maintain the speed without sacrificing software quality: (i) complexity-based routing of work across human and AI executors, and (ii) structured vibe-coder-to-developer handoff for high-risk work.

(i) Full-Stack Routing by Complexity

A structured execution plan is created at the start of each development cycle. Each task is assigned to one of three complexity tiers, which determines the execution path:

Tier	Description	Execution Path
A. High-complexity / high-risk	New logic, non-standard edge cases, critical security paths, or complex architectural decisions.	Full-stack engineer leads; AI assists with tooling, suggestions, and validation.
B. Medium-to-High complexity	Well-specified features with non-trivial integrations, data dependencies, or multi-system workflows.	Full-stack engineer defines instructions and guardrails; vibe-coder executes under specification.
C. Low-to-Medium complexity / standard patterns	Refactors, routine features, CRUD workflows, UI layers, and boilerplate patterns.	Vibe-coder executes autonomously.

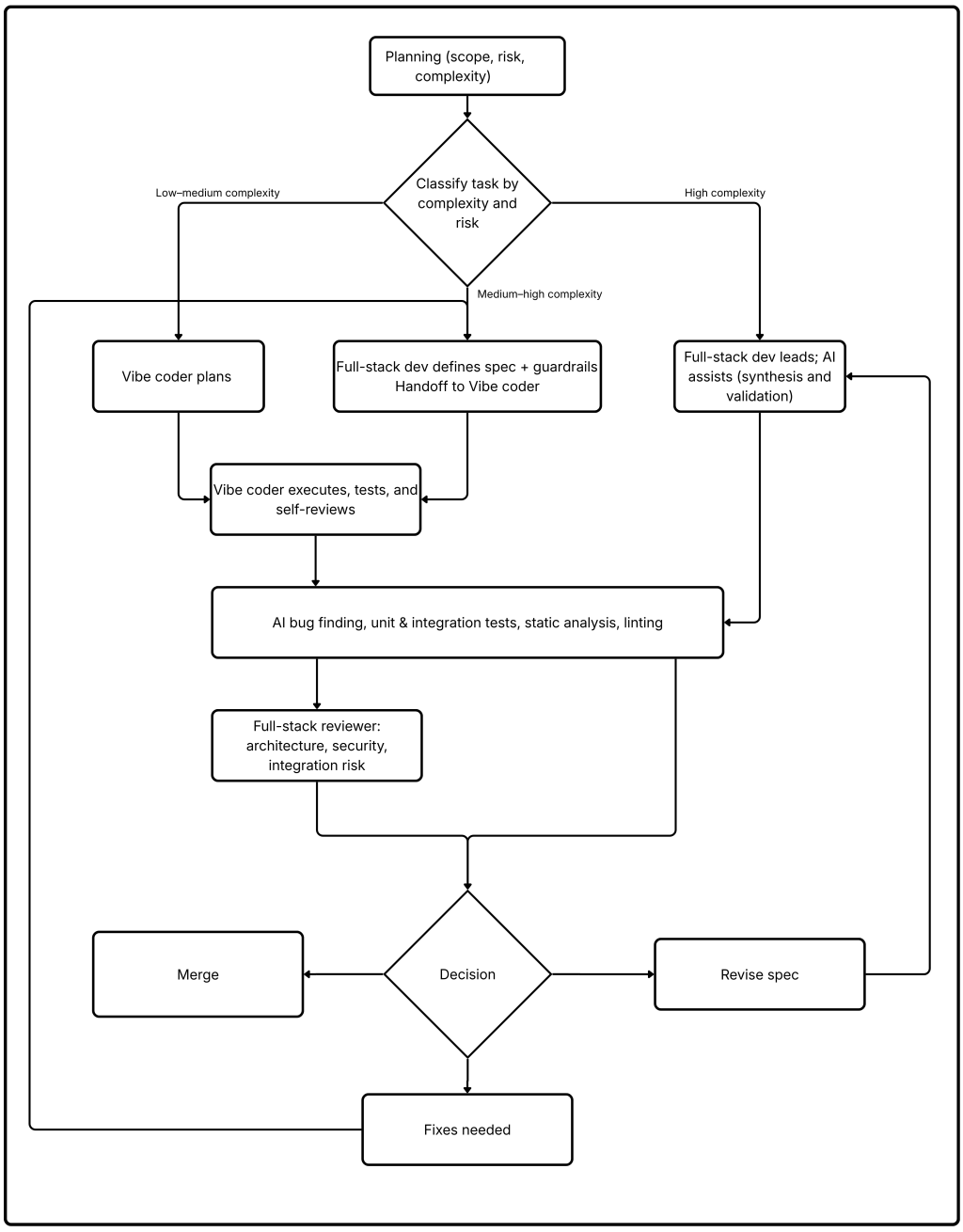
The tiering function maintains system stability. It ensures that scope is intentional, allocates tasks to the appropriate "executor" (whether vibe-coder, or Full-stack), and prevents the excessive or insufficient utilization of automation.

(ii) Systematic Hand-off: for High-Risk Tasks

When the vibe-coder performs Tier B or C tasks, the deliverables are returned via a systematic handoff procedure and have to pass AI-base testing workflows. The senior developer's role transitions from implementation to an expert reviewer with the following primary contributions:

- Architectural validation: resilience, scalability, and the impacts on data and infrastructure
- Security and accuracy: migrations, essential code pathways, and authentication boundaries
- Different approaches to problem-solving: eliminating model-induced impasses and clarifying ambiguous requirements.
- Integration risk: dependence challenges and implications for the entire system

• Planning & complexity routing



Results:

The vibe-coder executed approximately 80–90% of the implementation tasks

whereas full-stack developers concentrated on the critical 10–20%. Rather than addressing all tasks simultaneously, the full stack engineering used selected judgment, monitored the architecture, and resolved critical issues. This separation expedited processes and enhanced comprehension while maintaining accuracy in complex domains. We do not claim universal applicability; rather, this reflects empirical findings specific to our setting that could potentially be replicated in other settings.

4. Evaluation: Naturalistic Case Study

In accordance with Design Science evaluation practices (Venable et al., 2016), we assess our architecture via naturalistic observation of production development. We employed a single-case instrumental case study methodology (Stake, 1995) due to the following reasons: (1) the naturalistic setting effectively captures real constraints, (2) the rich instrumentation facilitates pattern detection across 3,676 sessions, and (3) our research objective focuses on theory-building, specifically generating testable hypotheses rather than testing existing theories.

This evaluation is structured around three research questions:

- Research Question (Feasibility): Can the two-layer context architecture support production-scale AI-assisted development in a real-world setting?
- Research Question 2 (Velocity Indicators): What observable patterns in development throughput and context efficiency emerge when systematic context engineering is applied over a 15-week project?
- Research Question 3 (Metrics): What methods can be employed to operationalize the measurement of context quality and AI effectiveness?

The standard design science trade off indicates that while a single-case study restricts external validity, it allows for greater depth of analysis. We address this issue through comprehensive documentation (facilitating replication), and clearly defined boundary conditions (Section 5.2). We analyzed our development process using quantitative git and IDE-level telemetry to investigate the correlation between systematic context patterns and code retention metrics. The datasets cover the period from July to October 2025.

Git metrics definitions.

To quantify development activity, we computed daily metrics from commit diffs on the main branch. Each metric was computed on a per-day basis and aggregated weekly and monthly. **Code Throughput (CT):** a measure of daily productive code activity, defined as the net lines of code added minus deleted. **Code Efficiency (CE):** the proportion of retained code relative to total code churn (adds + deletes), representing stability and reuse.

For calendar day d:

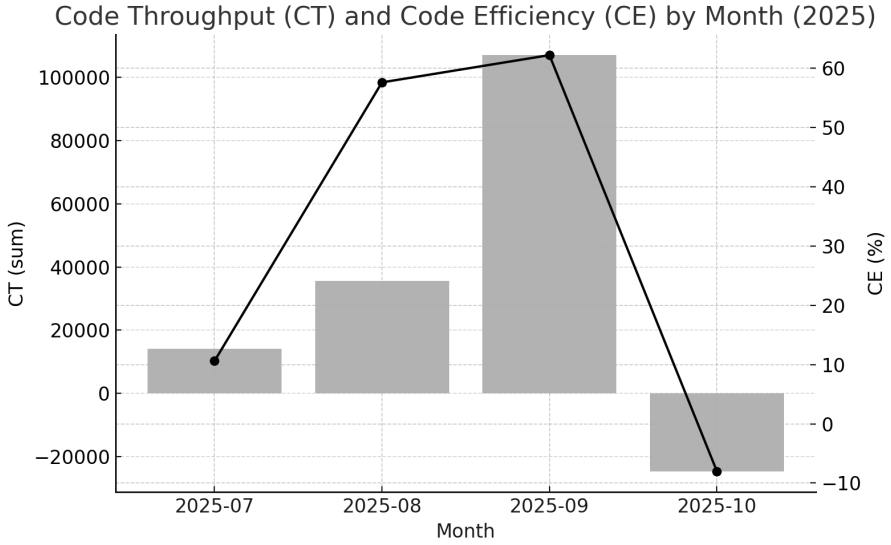
- Code Throughput $(CT_d) = LOC\_added_d - LOC\_deleted_d$
- Code Efficiency $(CE_d) = \frac{LOC\_added_d - LOC\_deleted_d}{LOC\_added_d + LOC\_deleted_d}$

with (CE_d) undefined when $LOC\_added_d + LOC\_deleted_d = 0$. Weekly buckets end on Sunday.

Period aggregates use **ratio-of-sums**:

$(CE_P) = \frac{CT_d}{(LOC\_added_d + LOC\_deleted_d)}$ $(CT_d) = CT_d$

Monthly $\left(\underline{CT}_{day}\right)$is the calendar-day mean. Missing days enter CT as zeros; CE is excluded when the denominator is zero. Winsorization (1st/99th pct) is used only for robustness plots. These operational definitions address RQ3 by establishing reproducible metrics for context quality (CRE) and development efficiency (CE, CT), enabling the pattern analysis in RQ2. The following chart summarizes monthly aggregates of CT and CE:



The pattern reflects July churn, an Aug–Sep buildout with rising retained work, and an October planned refactor.

Cursor telemetry.

IDE telemetry was exported via Cursor logs, capturing session-level token counts and cache events. From 3 988 IDE records, after filtering:

- Total Tokens > 0
- InputTokens ≥ 50

3 676 sessions remained.

Cursor Telemetry Definitions:
Generative Amplification (GA): the ratio of model-generated output tokens to input tokens, capturing generative expansion. **Context Reuse Efficiency (CRE):** the fraction of cached retrievals relative to total tokens, a proxy metric that reflects the extent of context reuse.

$(InputTokens)= Input(w/\ cache) + Input(w/o\ cache)$
Session Metrics:

- Generative Amplification $(GA_s)= \frac{Output\ Tokens}{Input\ Tokens}$,
- Context Reuse Efficiency $(CRE_s)= \frac{Cache\ Read}{Total\ \ Tokens}$

We report (i) medians (GA winsorized at 1/99; CRE raw) and (ii) ratio-of-sums per period p:

$(GA_P)= \frac{Output\ Tokens}{Input\ Tokens}$, $(CRE_P)= \frac{Cache\ Reads}{Total\ \ Tokens}$

Tables reports GA and CRE statistics:

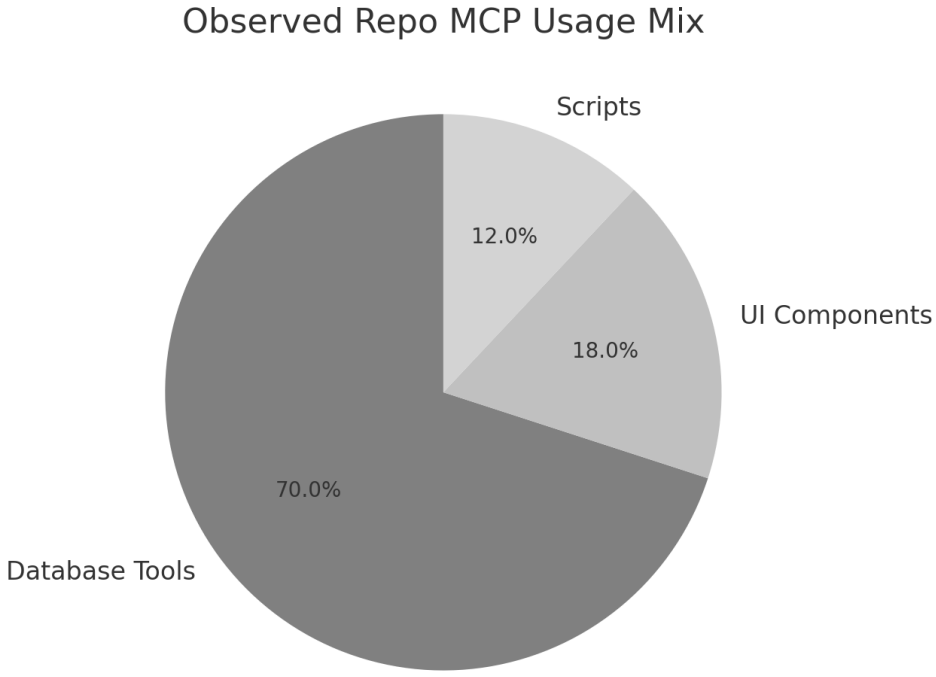
	Medians (winsorized GA): (GA_s) **- Generative Amplification**	(CRE_s) **- Context Reuse Efficiency**
Jul	0.0775	0.9234
Aug	0.0998	0.9414
Sep	0.0885	0.9593
Oct	0.0761	0.9377

	Monthly ratio-of-sums (GA_P) **- Generative Amplification**	(CRE_P) **- Context Reuse Efficiency**
Jul	0.0536	0.9009
Aug	0.0474	0.9088
Sep	0.0460	0.9223
Oct	0.0574	0.9146

Overall (ratio-of-sums across all sessions): (GA) 0.0497, (CRE) 0.9108; share $(CRE_s > 0.2)$=0.9587.

Linkage and Native Repo MCP. During the Aug–Sep buildout, weekly Code Efficiency (CE) rose alongside consistently high Context Reuse Efficiency (CRE), while Generative Amplification (GA) remained within a narrow band. This pattern supports and is consistent with our hypothesis that the framework: MCP-assisted retrieval reduces prompt/context debt, improving code retention without inflating token output.

Empirical telemetry confirms this dynamic. As the Repo MCP's rule-based context injection matured, (CRE) increased from 0.901 to 0.922 (ratio-of-sums) while (GA) declined from 0.054 to 0.046, reflecting a shift from generative exploration toward retrieval-augmented refinement. October's negative $(CT)/(CE)$ values correspond to a planned refactoring phase, showing that structural cleanup still manifests as negative productivity under line-based metrics despite greater contextual efficiency.



Database tools dominate (~70%), followed by UI components (~18%) and scripts (~12%), aligning with retrieval-heavy reuse and template grounding rather than ad-hoc prompting.

4.1 System Scope and Feature Complexity Internal Validity. Task heterogeneity across development phases could independently drive the observed trends. An early exploration naturally involves broader code generation, while later refinement focuses on targeted modifications. This shift might explain CRE increases independent of framework effects. To partially address this, we report both session-level medians and period ratio-of-sums, finding consistent patterns across measures. However, this does not eliminate the fundamental limitation that task heterogeneity and project phase naturally influence these metrics independent of any framework effects.

External Validity and Limitation. Results derive from a single event-tech SaaS platform built by a two-person team with specific complementary skills. Generalization remains unvalidated across: (1) other domains (embedded systems, infrastructure, data engineering), (2) different team sizes or skill distributions, and (3) alternative technology stacks. The "zero-friction stack" was deliberately selected for AI-friendliness; teams using legacy systems, microservices architectures, or statically-compiled languages may encounter additional friction that the framework does not address.

Construct Validity. CRE measures cache token reuse at the IDE boundary but does not directly capture code quality or comprehension depth. High CRE could reflect repetitive low-value tasks rather than effective knowledge retrieval. However, the observed correlation between rising CRE and sustained positive CE (57.6%→62.2% during Aug-Sep) suggests reuse patterns align with productive output during stable build phases. Lines-of-code remains a coarse metric that cannot distinguish algorithmic complexity from boilerplate generation.

Conclusion Validity. The single-project, 15-week timeline limits statistical power to descriptive analysis rather than inferential claims. October's negative CE (-8.0%) from planned refactoring represents 25% of the observation period showing framework limits under technical debt reduction. This is a lifecycle-normal phase but note that sustained velocity requires periodic refactoring that temporarily reverses productivity metrics. Telemetry filtering excluded 312 sessions (7.8%) based on InputTokens ≥ 50; the choice of this threshold was pragmatic but not systematically validated.

4.2 Evidence of Feasibility: research artifact as a Production System (RQ1) RQ1 asks whether the two-layer context architecture can support production-scale AI-assisted development in a real-world setting. The scale and complexity of research artifact provides positive evidence for at least one context. The system serves as a practical validation of the framework, functioning as a case study for developing a sophisticated, production-ready system within a competitive market.

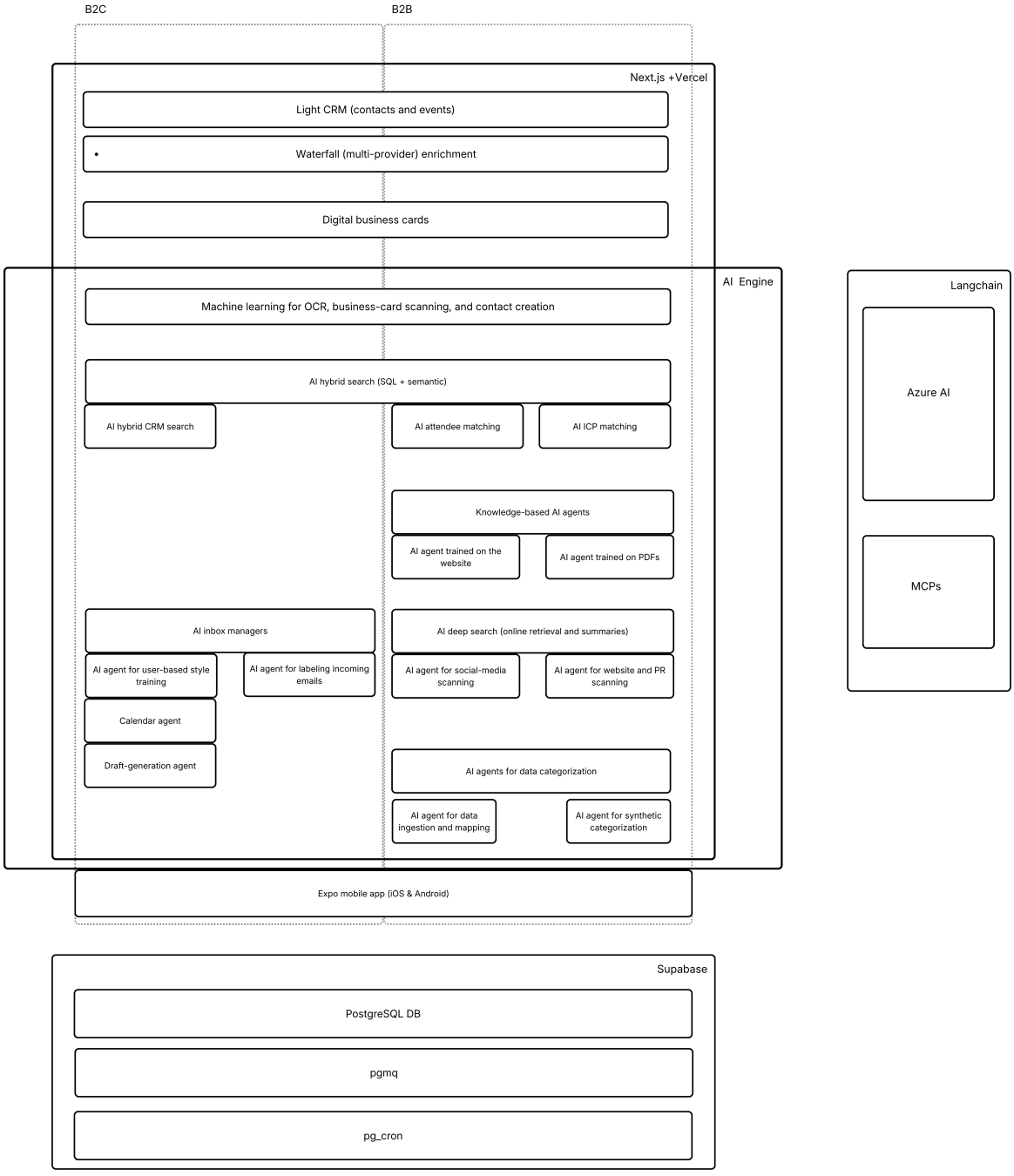
The event-productivity tooling sector provides an appropriate benchmark for evaluating artifact's scope and functional parity. Comparable platforms such as Popl ~$2M raised in 2021 (*Tracxn*, 2025) which reported a $4.4M qualified pipeline generated in Q2 of 2025 (*POPL Case Study*, 2025) or Blinq 25M raised (*Blinq Raises US$25m Series a to Reimagine the Start of Every Professional Relationship*, 2025); Clay.earth $8M raised (Clay, 2021), and acquired by Automattic in 2025 (Clay, 2025), and Grip ($14.5M raised) (Tracxn, 2025), Fyxer AI ($43M raised) *(Tracxn*, 2025b) - have each required substantial investment and extended development cycles to reach functional maturity. In approximately 15 weeks, our two-person part-time team produced a system that, by our subjective assessment, approximately 70% of the core functionality found in the above-mentioned platforms (Appendix B). While these established firms maintain

broader feature sets and long-standing market distribution channels, the results suggest that, through the proposed approach, comparable technical outcomes can be reproduced with a fraction of the R&D expenditure and time. Unlike legacy platforms that retrofit AI features, the research artifact was designed from inception around an AI-native architecture and we observed that the same infrastructure yields a more modular and easily extensible system. In addition to the fundamental SaaS capabilities, research artifact incorporates a multi-layer AI engine, demonstrating how the same design principles can be applied for the creation of AI-native workflow.

- **Agentic Prospect Qualification:** An autonomous AI-workflow that qualifies leads against Ideal Customer Profiles (ICPs).
- **Pre-Meeting Intelligence:** Agents actively scan the web and social media for key news and insights on prospects' companies.
- **Hybrid Networking Engine:** Merges the precision of SQL query with the flexibility of semantic search for superior connection discovery.
- **Conversational Knowledge Hub:** A dynamic RAG system, powered by Langchain and embedding models, that turns documents and websites into an interactive, chat-based knowledge base for events.
- **AI-Powered Data Structuring:** Utilizes small and efficient models like GPT5-mini to automatically categorize and structure raw data for the analytics and ingestion engine.
- **Personalized Inbox Orchestration:** Centralizes inbound email, learns user-specific labeling schemes, and automatically triages and tags high-signal threads.
- **Style-Adaptive Draft Generation:** Trains on the user's historical communication to generate reply drafts that preserve tone, structure.
- **Calendar-Aware Scheduling Agent:** Reads the user's calendar to automatically propose available time slots and respond to scheduling requests with precise availability.

The architecture was designed for incremental extension through modular AI services, MCPs and LangGraph implementation-ready. The research artifact serves as an existence proof that this approach can work in at least one context, though broader validation is needed. This delivery scope and timeline directly addresses RQ1's feasibility question, and observes that in one context, a small, focused team with appropriate skills and tools can achieve rapid development of a feature-rich AI native system, though long-term maintenance, scaling to large

user bases, and achieving product-market fit remain separate challenges beyond the scope of this case study.



5. Discussion, Limitations, and Future Directions

5.1 Design Knowledge: Generalizable Principles

Following Wieringa's (2014) framework of design knowledge, we extract generalizable principles from the implementation. These principles describe patterns, underlying mechanisms, and scope conditions for AI-assisted software development workflows.

Principle 1: Layered Context Architecture

Pattern. Separate stable declarative constraints (e.g., architectural rules, invariants) from dynamic project state (e.g., schemas, components, migrations) into distinct context layers.

Mechanism. Declarative rules change slowly, while implementation artifacts evolve continuously. Treating them as a single, undifferentiated context surface increases maintenance cost and raises the risk of exposing stale or contradictory information to the model. A layered design supports coherent evolution: rules define what must remain true, state reflects what is currently implemented.

Scope and limits. This principle is most effective in projects with durable architectural decisions and frequent code changes, such as long-lived production systems. In minimal or short-lived prototypes, the overhead of constructing separate layers may not be justified.

Principle 2: MCP-Mediated Retrieval

Pattern. Expose project structure through a queryable interface (e.g., an MCP server) so that agents retrieve context on demand, rather than relying primarily on large static injections into the prompt.

Mechanism. On-demand retrieval reduces lost-in-the-middle effects by allowing the model to pull specific artifacts (files, schemas, configurations) at the moment they are needed, instead of competing for a fixed context window. This shifts the design problem from "what to preload" to "how to address and fetch," which is more stable as the codebase grows.

Scope and limits. MCP-mediated retrieval is particularly valuable in codebases with non-trivial dependency graphs and frequent cross-module interaction, where full-project dumps are either infeasible or ambiguous. It introduces an additional dependency on server-side retrieval infrastructure, which may be unnecessary for very small or flat projects.

Principle 3: Co-Design of Context and Tools

Pattern. Design the context architecture jointly with the capabilities and constraints of the concrete tools in use (IDE indexing, MCP features, cache behavior, model endpoints), rather than treating tools as interchangeable implementation details.

Mechanism. The features of the tool determine which retrieval, navigation, and execution patterns are feasible in practice. An architecture that aligns with the indexing model of the IDE, the MCP resource, and the caching semantics of the environment can exploit those advantages to achieve more reliable and predictable agent behavior. Purely abstract designs that ignore these specifics tend to degrade in real use.

Scope and limits. This principle favors ecosystems where the toolchain is relatively stable (e.g., a team standardized on a given IDE and model provider). It entails a tradeoff: stronger local optimization at the cost of reduced portability if the tools change.

Principle 4: Amortization of Upfront Investment

Pattern. Treat context engineering as a high fixed-cost, low marginal-cost investment that must be amortized across many AI-assisted sessions and iterations.

Mechanism. Constructing rule layers, MCP endpoints, and retrieval workflows requires substantial initial effort. Once established, each additional interaction benefits from reduced cognitive overhead, faster navigation, and more reliable agent outputs. The payoff emerges not from any single session, but from repeated reuse across the project lifecycle.

Scope and limits. This principle is appropriate for multi-sprint or multi-month initiatives, where the fixed cost can be amortized over sustained usage. For MVPs or experiments constrained to a few weeks, heavy context engineering is unlikely to recoup its upfront cost and lighter-weight workflows may be preferable.

5.2 Discussion and conclusion

A persistent execution gap separates optimistic claims about AI-assisted development from reproducible, system-level outcomes. This work suggested that sustained speed emerges when workflow, architecture, and validation are explicitly engineered rather than left implicit. Our experience suggests that the execution gap may stem partly from methodological challenges, though determining relative contributions requires controlled experiments.

Our three research questions guide the analysis. RQ1 examines whether the proposed architecture supports production-scale development in an applied setting. RQ2 analyzes session-level behavioral signatures over a 15-week cycle, and RQ3 evaluates whether these patterns can be captured through reproducible git and IDE telemetry. Together, these results show initial feasibility and measurable effects, although broader generalizability remains an open question.

Following Design Science Research methodology, the case study demonstrates empirically that when context is industrialized. As the declarative rulebook stabilized and a programmatic context surface (MCP server integrated within an AI-native IDE) exposed authoritative schemas, components, and scripts, we observed a consistent pattern: cache reuse increased while generative expansion

decreased over time. The evidence is consistent with a transition from open-ended drafting to targeted, verifiable modification, linking concrete context mechanisms to measurable behavior at session scale. The proposed approach for high velocity development rests on four mutually reinforcing pillars:

- **A low-overhead, AI-optimized development environment.** Components are selected to minimize integration overhead, support rapid reversibility, and provide strong documentation and LLM affordances for fast API exploration and debugging.
- **High-Discipline Operations**. Planning is separated from execution; risky actions are gated; and post-feature maintenance removes deprecated code and reduces retrieval noise for subsequent work.
- **Systematic Context Engineering.** Non-negotiable constraints are encoded as declarative rules, while an MCP server programmatically exposes live structure so agents and IDEs query ground truth rather than infer it.
- **Agentic Orchestration.** Work is decomposed into role-based agents with explicit handoffs for analysis, implementation, and validation; long chains run in queues; completion is bound to verification artifacts.

The system under study implements these pillars fully. A two-person, part-time team delivered an enterprise-grade, multi-feature SaaS platform of roughly 220k lines of production code in fifteen weeks, implementing an estimated 70 percent of the functionality found in established platforms in this domain. While the case demonstrates feasibility and reveals consistent behavioral patterns under these conditions, we do not assert claims about production scalability, long-term maintenance properties, or applicability to different teams or domains.

Conceptually, if AI-assisted development continues to mature, competitive advantage may depend less on engineering headcount than on methodological rigor: the ability to encode knowledge, expose authoritative structure, and maintain auditability across iterations. This hypothesis warrants further empirical testing across varied organizational contexts.

References

Agents | Playwright. (2025). https://playwright.dev/docs/test-agents

Announcing Clay's seed round. (2021). Clay Earth.https://clay.earth/usecase/seed-announcement

Anthropic. (2024). Model Context Protocol. Retrieved from https://modelcontextprotocol.io

Anthropic Engineering. (2025, September 28). *Effective context engineering for AI agents*. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

Axios. (2025, September 18). *Anthropic's Dario Amodei & Jack Clark & Axios' Jim VandeHei* [Video]. YouTube. https://www.youtube.com/watch?v=nvXj4HTiYqA

Barla, N. (2025, August 4). *Context rot: Why LLMs are getting dumber?* Adaline Labs. https://labs.adaline.ai/p/context-rot-why-llms-are-getting

Beck, K. (2000). *Extreme programming explained: Embrace Change*. Addison-Wesley Professional.

Becker, J., Rush, N., Barnes, E., & Rein, D. (2025, July 12). *Measuring the impact of early-2025 AI on experienced open-source developer productivity*. arXiv.org. https://arxiv.org/abs/2507.09089

Bertsch, A., Ivgi, M., Xiao, E., Alon, U., Berant, J., Gormley, M. R., & Neubig, G. (2025, March 5). *In-Context Learning with Long-Context Models: An In-Depth Exploration*. arXiv.org. https://arxiv.org/abs/2405.00200

Baek, J., Lee, S. J., Gupta, P., Oh, G., Dalmia, S., & Kolhar, P. (2025, May 28). Revisiting In-Context Learning with Long Context Language Models. arXiv.org. https://arxiv.org/abs/2412.16926

Blinq raises US$25m Series A to reimagine the start of every professional relationship. (2025). https://blinq.me/blog/blinq-raises-us-25m-series-a-to-reimagine-the-start-of-every-professional-relationship

Bort, J. (2025, June 20). 6-month-old, solo-owned vibe coder Base44 sells to Wix for $80M cash. *TechCrunch*. https://techcrunch.com/2025/06/18/6-month-old-solo-owned-vibe-coder-base44-sells-to-wix-for-80m-cash/

Bort, J. (2025b, September 10). Replit hits $3B valuation on $150M annualized revenue. *TechCrunch*. https://techcrunch.com/2025/09/10/replit-hits-3b-valuation-on-150m-annualized-revenue/

Claude Docs. (2025). Claude Docs. https://docs.claude.com/en/home

Clay. (2025). *Clay x Automattic*. Clay. https://clay.earth/next

Code execution with MCP: building more efficient AI agents. Antropic (2025). https://www.anthropic.com/engineering/code-execution-with-mcp

Context ROT: How increasing input tokens impacts LLM performance. (2025, July 13). Chroma Research. https://research.trychroma.com/context-rot

Cursor. (2025, September 29). *Cursor: The best way to code with AI*. https://cursor.com/

Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2025). Unleashing the potential of prompt engineering for large language models. *Patterns*, 101260. https:

//doi.org/10.1016/j.patter.2025.101260

Du, Y., Tian, M., Ronanki, S., Rongali, S., Bodapati, S., Galstyan, A., Wells, A., Schwartz, R., Huerta, E. A., & Peng, H. (2025). Context length alone hurts LLM performance despite perfect retrieval. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2510.05381

Expo for mobile apps - Bolt. (2025). Bolt. https://support.bolt.new/integrations/expo

Find your Template. (2025). Vercel. https://vercel.com/templates

From idea to app with Replit and Expo. (2025). Expo Blog. https://expo.dev/blog/from-idea-to-app-with-replit-and-expo

Gemini models. (2025). Google AI for Developers. https://ai.google.dev/gemini-api/docs/models

Gregor, S., & Hevner, A. R. (2013). Positioning and presenting design science research for maximum impact1. MIS Quarterly, 37(2), 337–355. https://doi.org/10.25300/misq/2013/37.2.01

Grip has raised $13 Million to create the first Market Engagement Platform - Grip Events News. (2021).https://www.grip.events/news/grip-has-raised-13-million-to-create-the-first-market-engagement-platform

GitHub. (2025, August 22). GitHub - github/spec-kit: *Toolkit to help you get started with spec-driven development*. GitHub. https://github.com/github/spec-kit

Heineman, D., Hofmann, V., Magnusson, I., Gu, Y., Smith, N. A., Hajishirzi, H., Lo, K., & Dodge, J. (2025, August 18). Signal and noise: *A framework for reducing uncertainty in language model evaluation*. arXiv.org. https://arxiv.org/abs/2508.13144

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1), 75–105. https://doi.org/10.2307/25148625

Introducing GPT-5 for developers. (2025). OpenAI. https://openai.com/index/introducing-gpt-5-for-developers/

Kavukcuoglu, K. (2025, March 28). *Gemini 2.5: Our most intelligent AI model*. Google. https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/

Lakera Website. (2025). *LLM hallucinations in 2025: How to understand and tackle AI's most persistent quirk* | Lakera – Protecting AI teams that disrupt the world. https://www.lakera.ai/blog/guide-to-hallucinations-in-large-language-models

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023, July 6). *Lost in the middle: How language models use long contexts*. arXiv.org. https://arxiv.org/abs/2307.03172

LLM Context Management: How to improve performance and lower costs. (2025). 16x Eval. https://eval.16x.engineer/blog/llm-context-management-guide

Khan, I., Chowdary, A., Haseeb, S., Patel, U., & Zaii, Y. (2025, July 14). *Kodezi Chronos: A Debugging-First Language Model for Repository-Scale Code Understanding*. arXiv.org. https://arxiv.org/abs/2507.12482

Ko, A. J., Myers, B. A., & Aung, H. H. (2018). Six Learning Barriers in End-User Programming Systems (Version 1). Carnegie Mellon University. https://doi.org/10.1184/R1/6470432.v1

Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z., Zhang, D., Zhou, C., Mao, J., Xia, T., Guo, J., & Liu, S. (2025, July 17). *A survey of context engineering for large language models*. arXiv.org. https://arxiv.org/abs/2507.13334

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science research Methodology for Information Systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/mis0742-1222240302

POPL case study. (2025). Popl Digital Business Card. https://popl.co/pages/popl-case-study

Razzaq, A. (2025, August 20). *Signal and noise: Unlocking reliable LLM evaluation for better AI decisions*. MarkTechPost. https://www.marktechpost.com/2025/08/20/signal-and-noise-unlocking-reliable-llm-evaluation-for-better-ai-decisions/

Reduce Context Rot in LLMs with RAG & DeepSearch | Valyu | Valyu. (2025, August 19). Valyu. https://www.valyu.ai/blogs/reduce-your-ai-agents-context-rot-with-search-apis-and-rag

Salva, R. J. (2025, September 23). How are developers using AI? Inside our 2025 DORA report. Google. https://blog.google/technology/developers/dora-report-2025/

SECRePoBench: Benchmarking LLMs for secure code Generation in Real-World Repositories. (2025). https://arxiv.org/html/2504.21205v1

Shaw, M. (1996). Some patterns for software architectures (pp. 255–269). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.674

Stake, R. E. (1995). The art of case study research. SAGE.

Supabase Integration - lovable documentation. (2025). Lovable Documentation. https://docs.lovable.dev/integrations/supabase

Stanford HAI. (2025, March). *AI Index Report 2025*. Retrieved from https://hai.stanford.edu

The 2025 AI Index Report | Stanford HAI. (2025). https://hai.stanford.edu/ai-index/2025-ai-index-report

Tracxn. (2025). Popl - Company Profile. https://tracxn.com/d/companies/popl/___bU61QF23Sj4m1Zb5AEue6qP_eobCLPpElnjbBCSS7dY

Tracxn. (2025b). Fyxer - Company Profile. https://tracxn.com/d/companies/fyxer/___4WJDNopySFY-UYw-Ky1DM5as9muOSwhgKv-TGSYqaYQ

Temkin, M. (2025, June 5). Cursor's Anysphere nabs $9.9B valuation, soars past $500M ARR. *TechCrunch*. https://techcrunch.com/2025/06/05/cursors-anysphere-nabs-9-9b-valuation-soars-past-500m-arr/

Venable, J., Pries-Heje, J., & Baskerville, R. (2014). FEDS: a Framework for Evaluation in Design Science Research. *European Journal of Information Systems*, *25*(1), 77–89. https://doi.org/10.1057/ejis.2014.36

Visma develops new code up to 50 percent faster with GitHub Copilot and Azure DevOps | Microsoft Customer Stories. (2025). https://www.microsoft.com/en/customers/story/1774868194783832907-visma-visual-studio-professional-services-en-norway#:~:text=Even%20those%20working%20on%20projects,expanded%20revenue%20and%20market%20share

Vilakati, S. (2025). Prompt engineering for accurate statistical reasoning with large language models in medical research. *Frontiers in Artificial Intelligence*, *8*. https://doi.org/10.3389/frai.2025.1658316

Wang, M., Kojima, T., Iwasawa, Y., & Matsuo, Y. (2025). *Lost in the distance: Large language models struggle to capture long-distance relational knowledge.* Findings of the Association for Computational Linguistics: NAACL 2025, 4536–4544. https://doi.org/10.18653/v1/2025.findings-naacl.256

Wei, Z., Wang, S., Rong, X., Liu, X., & Li, H. (2025, May 22). *Shadows in the attention: Contextual perturbation and representation drift in the dynamics of hallucination in LLMs.* arXiv.org. https://arxiv.org/abs/2505.16894

What is the Model Context Protocol (MCP)? - *Model Context Protocol.* (2025). Model Context Protocol. https://modelcontextprotocol.io/docs/getting-started/intro#:~:text=,on%20your%20behalf%20when%20necessary

Wieringa, R. (2014). *Design science methodology for information systems and software engineering*. Springer-Verlag Berlin Heidelberg.

$100M ARR & lovable agent. (2025, October 20). Lovable. https://lovable.dev/blog/agent