The Agentic Governance Framework: Operationalizing Vibe Coding for Production

1. Executive Summary: Solving "Vibe Collapse" Once and For All

The software industry is currently navigating a transition from "Vibe Coding"—a chaotic, prompt-driven prototyping method—to **Agentic Engineering**, a disciplined practice capable of delivering production-grade software. While Vibe Coding offers exponential speed for "0 to 1" development, it inherently suffers from "Vibe Collapse": the point where a project's complexity exceeds the AI's context window and reasoning capacity, leading to unmanageable technical debt and security vulnerabilities.

To make Vibe Coding "truly work" for revenue-generating products, organizations must move beyond simple prompt engineering to **Flow Engineering** and **Spec-Driven Development (SDD)**. This report outlines the **Agentic Governance Framework**, a comprehensive system designed to mitigate hallucinations, enforce architectural standards, and ensure security "one time and always." By shifting the burden of discipline from human memory to automated agentic workflows, developers can harness AI's speed without sacrificing long-term maintainability.

---

2. The Core Shift: From Prompting to Flow Engineering

The primary failure mode of Vibe Coding is relying on a single, linear context window to hold all project knowledge. To solve this, we must adopt **Flow Engineering**—the practice of breaking complex coding tasks into structured, iterative workflows where AI agents interact with defined tools and quality gates.

2.1 The Flow Engineering Hierarchy

Unlike Prompt Engineering, which focuses on the quality of a single query, Flow Engineering focuses on the *architecture of the interaction*.

- **Prompt Engineering:** "Write a secure login function." (Prone to hallucinations and context loss).
- **Flow Engineering:** A state-machine approach where the AI must pass specific stages: **Spec -> Plan -> Test -> Implement -> Verify**. If any stage fails, the flow loops back automatically.

Benefit: This solves the "consistency" issue permanently by forcing every line of code to pass through the same rigorous, automated checkpoints, regardless of the developer's fatigue level or memory.

---

3. Pillar I: The Constitution (Context Engineering)

To solve coding standard issues "one time," you must codify them into a persistent context layer that the AI cannot ignore. This is often achieved through a project-specific "constitution," such as the .cursorrules file in the Cursor IDE or system prompts in agentic frameworks.[1]

3.1 The .cursorrules Governance File

This file acts as a persistent supervisor agent. Once configured, it enforces rules on every single prompt, effectively "fine-tuning" the model for your specific project constraints.

Critical Configuration Rules for Production:

1. **Tech Stack Strictness:** Explicitly forbid valid but unwanted patterns.
    - *Rule:* "Always use strict TypeScript. Do NOT use any. Use zod for all schema validation.".[3]
2. **Behavioral Constraints:** Enforce maintenance habits.
    - *Rule:* "Never remove existing comments. Never leave 'TODO' placeholders. Update

documentation with every code change.".[5]

3. **Security Gates:** Hard-code security into the generation process.
    - *Rule:* "Never output real API keys. Use process.env. Sanitize all SQL inputs.".[6]

3.2 Spec-Driven Development (SDD)

Before a single line of code is written, the framework enforces the creation of a specification. Tools like spec-kit allow developers to generate a spec.md (What are we building?), plan.md (How will we build it?), and tasks.md (Atomic steps).

- **The Fix:** This prevents "coding into a corner" by forcing the AI to "think" architecturally before it commits to implementation.

---

4. Pillar II: The Immune System (AI-TDD)

The most effective way to solve logic errors and hallucinations is to invert the coding process using **AI-Augmented Test-Driven Development (AI-TDD)**. In this workflow, tests are not an afterthought; they are the requirement specification.

4.1 The Red-Green-Refactor Loop

1. **Red (The Constraint):** The AI is prompted to write a *failing test* that asserts the desired behavior. This test serves as the "truth" for the feature.
2. **Green (The Solution):** The AI generates the implementation code. Its only goal is to pass the test. If it hallucinates a method, the test runner immediately rejects it.
3. **Refactor (The Cleanup):** Once the test passes, the AI is tasked with optimizing the code. The test suite ensures no regressions occur during this phase.[7]

Automated "Circuit Breakers":
In advanced flows, if the AI fails to pass the test after 3 attempts, the workflow triggers a "Circuit Breaker," halting the process and requesting human intervention. This prevents the agent from spiraling into a hallucination loop where it writes increasingly bizarre code to satisfy a prompt.

---

5. Pillar III: Multi-Agent Architecture (The Digital Team)

To achieve "one time and always" reliability, we must move from a single "coder" agent to a system of specialized agents that check each other's work.[8]

5.1 The Role-Based Graph

Using frameworks like **LangGraph** or **AutoGen**, we can simulate a full engineering team [10]:

- **The Planner:** Breaks user requests into a dependency graph. Does not write code.
- **The Coder:** Executes the plan one file at a time.
- **The Critic (Reviewer):** An adversarial agent prompted to find security flaws and bugs. It rejects code that doesn't meet OWASP standards.[12]
- **The Integrator:** Ensures that new code doesn't break existing builds or circular dependencies.

5.2 Self-Correcting Workflows

This architecture enables "Self-Healing." If the Critic finds a bug, it feeds the error back to the Coder with specific instructions. This loop continues until the code passes strict quality gates, all without human intervention.

---

6. Pillar IV: Economic & Security Discipline

Making vibe coding "profitable" requires treating technical debt as a financial metric that must be managed proactively.

6.1 The Refactoring Budget

Allocate 10-20% of agent compute time specifically to "Refactoring Mode." In this mode, agents scan the codebase not to add features, but to reduce cyclomatic complexity and unify duplicate logic.[14]

- *Profitability Logic:* This "maintenance tax" prevents the exponential cost of changing a brittle codebase later.

6.2 The Production Readiness Checklist

Before any "vibe coded" prototype moves to production, it must pass a rigorous audit [16]:

1. **Supply Chain Audit:** Verify no hallucinated or malicious packages were installed.[18]
2. **Secrets Check:** Ensure no API keys are hardcoded (a common LLM bad habit).[19]
3. **Load Testing:** AI code often ignores performance. Automated agents should run stress tests (e.g., k6) to verify scalability.[20]
4. **OWASP Scan:** Automated checks for the Top 10 LLM vulnerabilities, including Prompt Injection and Insecure Output Handling.[18]

---

7. Implementation: The "One Time" Setup Guide

To implement this framework immediately, follow these steps to establish your governance layer.

Step 1: Initialize the Constitution

Create a .cursorrules (or equivalent) file in your project root. Paste in your non-negotiable standards.

> *Resource:* Use templates from the awesome-cursorrules repository to ensure

comprehensive coverage of TypeScript, React, and Security best practices.[21]

Step 2: Establish the TDD Loop

Adopt the strict rule: **"No Code Without a Failing Test."** Use an alias or macro in your IDE that prompts the AI to generate the test *first*.[23]

Step 3: Deploy Automated Critics

Integrate an AI code review tool like **CodeRabbit** or **Qodo** into your GitHub/GitLab workflow. These tools act as the "always-on" Reviewer agent, catching issues in every Pull Request before a human looks at it.[13]

Step 4: Automate the Circuit Breaker

Configure your agentic tools (e.g., inside Cursor or via LangGraph) to stop and ask for help if a task takes more than 5 iterations. This saves token costs and prevents "spaghetti code" generation.[25]

Conclusion

The transition from Vibe Coding to **Agentic Governance** is the difference between a toy and a business. By implementing **Flow Engineering**, **Context Engineering**, and **AI-TDD**, developers create a rigid framework that allows the "vibes" to flow safely. This system solves the problems of fragility and debt "one time" through configuration, ensuring that the solution remains effective "always" through automated enforcement.

Works cited

1. Show me your general prompt for Rules for AI from Cursor settings - Reddit,

accessed November 30, 2025, https://www.reddit.com/r/cursor/comments/1faf2rw/show_me_your_general_prompt_for_rules_for_ai_from/
2. .CursorRules Rules - Mastering AI-Assisted Coding: Unlock the Power of .cursorrules in Cursor IDE, accessed November 30, 2025, https://dotcursorrules.com/
3. My `.cursorrules` Configuration for Full-Stack TypeScript/Next.js Development, accessed November 30, 2025, https://dev.to/simplr_sh/my-cursorrules-configuration-for-typescriptnextjs-development-5ep7
4. The ultimate .cursorrules for TypeScript, React 19, Next.js 15, Vercel AI SDK, Shadcn UI, Radix UI, and Tailwind CSS : r/cursor - Reddit, accessed November 30, 2025, https://www.reddit.com/r/cursor/comments/1gjd96h/the_ultimate_cursorrules_for_typescript_react_19/
5. Getting Better Results from Cursor AI with Simple Rules | by Andi Ashari - Medium, accessed November 30, 2025, https://medium.com/@aashari/getting-better-results-from-cursor-ai-with-simple-rules-cbc87346ad88
6. Best Practices: .cursorrules - How To - Cursor - Community Forum, accessed November 30, 2025, https://forum.cursor.com/t/best-practices-cursorrules/41775
7. AAID: Augmented AI Development - DEV Community, accessed November 30, 2025, https://dev.to/dawiddahl/aaid-augmented-ai-development-50c9
8. Multi-Agent Workflows: A Practical Guide to Design, Tools, and Deployment - Medium, accessed November 30, 2025, https://medium.com/@kanerika/multi-agent-workflows-a-practical-guide-to-design-tools-and-deployment-3b0a2c46e389
9. Agentic Code Generation Papers Part 2 (Last), accessed November 30, 2025, https://cbarkinozer.medium.com/agentic-code-generation-papers-part-2-23d6482da032
10. Multi-Agent Code Review using Generative AI and LangGraph - YouTube, accessed November 30, 2025, https://www.youtube.com/watch?v=pdnT3yLk70c
11. awesome-cursor-rules-mdc/rules-mdc/autogen.mdc at main - GitHub, accessed November 30, 2025, https://github.com/sanjeed5/awesome-cursor-rules-mdc/blob/main/rules-mdc/autogen.mdc
12. Google CEO Sundar Pichai on how vibe Coding can help non-tech graduates build careers in technology, accessed November 30, 2025, https://timesofindia.indiatimes.com/technology/tech-news/google-ceo-sundar-pichai-on-how-vibe-coding-can-help-non-tech-graduates-build-careers-in-technology/articleshow/125631791.cms
13. AI Code Reviews | CodeRabbit | Try for Free, accessed November 30, 2025, https://www.coderabbit.ai/
14. AI code refactoring: Strategic approaches to enterprise software modernization in 2025 - DX, accessed November 30, 2025,

https://getdx.com/blog/enterprise-ai-refactoring-best-practices/
15. Don't Let GenAI Coding Become Technical Debt: 9 Pillars for ..., accessed November 30, 2025, https://jazmy.medium.com/dont-let-genai-coding-become-technical-debt-9-pillars-for-success-1074775d0fe3
16. 8 Production Readiness Checklist for Every AI Agent | Galileo, accessed November 30, 2025, https://galileo.ai/blog/production-readiness-checklist-ai-agent-reliability
17. From Prototype to Production: Scaling Your Smart Product, accessed November 30, 2025, https://www.mutuallyhuman.com/from-prototype-to-production/?utm_source=rss&utm_medium=rss&utm_campaign=from-prototype-to-production
18. OWASP Top 10 LLM Vulnerabilities & Security Checklist, accessed November 30, 2025, https://www.lasso.security/blog/owasp-top-10-llm-vulnerabilities-security-checklist
19. AI-Generated Code Security Checklist: 7 Policies Every CISO Needs - OpsMx, accessed November 30, 2025, https://www.opsmx.com/blog/ai-generated-code-security-checklist-7-policies-every-ciso-needs/
20. AI Code Review: What to Look For in the Age of Copilots - DEV Community, accessed November 30, 2025, https://dev.to/rakbro/ai-code-review-what-to-look-for-in-the-age-of-copilots-2g02
21. PatrickJS/awesome-cursorrules: Configuration files that enhance Cursor AI editor experience with custom rules and behaviors - GitHub, accessed November 30, 2025, https://github.com/PatrickJS/awesome-cursorrules
22. awesome-cursorrules/rules/web-app-optimization-cursorrules-prompt-file/.cursorrules at main · PatrickJS/awesome-cursorrules - GitHub, accessed November 30, 2025, https://github.com/PatrickJS/awesome-cursorrules/blob/main/rules/web-app-optimization-cursorrules-prompt-file/.cursorrules
23. AI-TDD: you write the test, GPT writes the code to pass it - DEV Community, accessed November 30, 2025, https://dev.to/disukharev/aitdd-ai-cli-for-tdd-you-write-the-test-ai-makes-it-green-32bn
24. 9 Best Automated Code Review Tools for Developers in 2025 - Qodo, accessed November 30, 2025, https://www.qodo.ai/blog/automated-code-review/
25. Build AI Agents That Self-Correct Until It's Right (ADK LoopAgent) | by Noble Ackerson | Google Developer Experts | Nov, 2025 | Medium, accessed November 30, 2025, https://medium.com/google-developer-experts/build-ai-agents-that-self-correct-until-its-right-adk-loopagent-f620bf351462