Hoyack SOFTWARE DEVELOPMENT

The Vibe Code Governance Playbook

How to Scale, Secure, and Sustain AI-Generated Code in Your Organization

Your team shipped faster than ever. Now what?

The Vibe Coding Revolution (And Its Hidden Costs)

Something shifted in the last year. Engineers who used to spend days on a feature are now shipping in hours. Cursor, ChatGPT, Codex, Claude—they've changed how code gets written. Product teams are seeing ideas come to life before the sprint ends. It feels like a superpower.

And it is. But it's also a trap.

Here's what we keep seeing: A prototype gets built in a weekend. It works. The demo goes well. Stakeholders get excited. Someone says "let's ship it." And suddenly that weekend project is in production, handling real customers, with code that was never architected, never reviewed for security, and definitely never designed to scale.

Six months later, that same team is drowning. Every new feature breaks something. Nobody understands half the codebase. The engineer who vibe-coded the original thing has moved on or forgotten how it works. And now there's a compliance audit coming.

Sound familiar?

The Three Stages of Vibe Code Debt

We've watched this play out at enough companies to see the pattern. It usually goes like this:

Stage	What It Looks Like	The Real Cost
The Prototype Trap	A vibe-coded MVP gets traction. Leadership says "scale it."	No tests. Hardcoded everything. Patterns that made sense to ChatGPT but not to your team.
The Maintenance Spiral	Features keep shipping, but everything takes longer. Bugs multiply.	Your best engineers are debugging instead of building. New hires take months to get productive.
The Reckoning	An audit. A security incident. A due diligence process.	You're facing a rebuild at the worst possible moment. And everyone's asking how it got this bad.

A Governance Framework That Actually Works

Look, we're not going to tell you to ban AI coding tools. That ship has sailed, and honestly, they're too useful to give up. The question isn't whether your team uses them. It's whether you have any idea what's happening with the code they produce.

Step 1: Figure Out What You're Dealing With

Before you can fix anything, you need to know what's out there. Go through your codebase and sort AI-generated code into three buckets:

Category	What It Is	What To Do
Throwaway	Demos, prototypes, internal scripts nobody relies on	Label it clearly. Keep it away from prod.
Transitional	It's in production, but it's held together with duct tape	Put it on the refactor list. Give someone ownership.
Critical Path	Touching revenue, customer data, or compliance	Stop what you're doing and review it now.

A Governance Framework That Actually Works

Step 2: Set Up Quality Gates

AI-generated code should go through the same checks as everything else. If you don't have these in place, start here:

- Minimum test coverage before anything gets merged
- Security scans that actually run (not ones everyone skips)
- Architecture review for anything touching data, auth, or external APIs
- Documentation requirements (even a few sentences explaining what the code does)

Step 3: Train Your Team to Spot AI Code Problems

AI-generated code fails in predictable ways. Your reviewers should know what to look for:

- Packages that don't exist or were deprecated years ago
- Patterns that work in tutorials but don't fit your stack
- Missing error handling (AI loves the happy path)
- 50 lines of code for something that should take 5
- Hardcoded secrets, SQL injection vulnerabilities, the classics

Refactor or Rebuild? How to Decide

Not everything needs to be thrown away. Sometimes a messy codebase can be cleaned up. Sometimes it can't. Here's how to tell:

Ask Yourself	You Can Refactor If...	You Probably Need to Rebuild If...
Is the architecture sound?	The bones are good, just messy	It's spaghetti all the way down
Can you add tests?	Piece by piece, yeah	You'd have to rewrite it to make it testable
Does anyone understand it?	At least one person can explain it	The person who built it left. Nobody knows.
How urgent is this?	You have time for gradual fixes	It's breaking now and needs to work yesterday
How much is broken?	Less than half	More than half

A Simple Vibe Code Policy

You don't need a 50-page document. You need clear guidelines that people will actually follow. Start with something like this:

Go Ahead (No Review Needed)	Needs a Second Set of Eyes	Don't Even Try It
• Internal scripts and automation • Prototypes that won't leave your laptop • Boilerplate and scaffolding • Docs and test generation	• Anything going to production • Code that touches customer data • Anything calling external APIs • Regulated environments (healthcare, finance, etc.)	• Security-critical code without an expert looking at it • Compliance workflows without audit trails • Core business logic that nobody reviews

The Real Problem: Integration

Here's what most people miss: the hard part isn't the code itself. It's connecting it to everything else.

AI tools are great at generating isolated solutions. But they have no idea about your:

- Existing data models
- Auth patterns
- Logging setup
- How you actually deploy things
- That weird vendor API you've been working around for years

This is where vibe-coded projects die. The prototype works on someone's laptop. Integrating it into your real environment breaks everything.

What we do at Hoyack

We take vibe-coded prototypes and make them production-ready. We connect them to your existing systems, add proper error handling, and make sure compliance isn't an afterthought. It's not glamorous work, but it's the work that actually matters.

Your Checklist

Print this out. Stick it on the wall. Work through it.

Action Item	Done?	Notes
Find all the AI-generated code in production	[ ]	
Sort it by risk (throwaway, transitional, critical)	[ ]	
Set up quality gates that people actually use	[ ]	
Train reviewers on AI-specific failure patterns	[ ]	
Write down a policy (even a short one)	[ ]	
Assign someone to own the transitional/critical stuff	[ ]	
Budget time/money to actually fix things	[ ]	
Map out what needs to integrate with what	[ ]	

Let's Talk

Vibe coding isn't going anywhere. It's too useful. But without some guardrails, it creates the kind of technical debt that sinks companies.

Brandon, our CEO, has spent the last year helping CTOs dig out of exactly this situation. Auditing codebases. Building governance frameworks. Refactoring the stuff that actually matters. If any of this sounds familiar, it might be worth a conversation.

Book a call. We'll figure it out together.

SCHEDULE A CONSULTATION

The Vibe Code Governance Playbook

How to Scale, Secure, and Sustain AI-Generated Code in Your Organization

Hoyack SOFTWARE DEVELOPMENT