SET THE UNIVERSAL CONSTANTS

How First-Principles Engineering Creates High-Velocity Organizations

By Chris Trahey

hello@christrahey.com · +1 (541) 390-9463

Author's Note: The Premise

Software organizations rarely fall apart because they lack information, capital, or raw talent. They fall apart because they lack coherence.

Every single day, executive teams and engineering leads gather in boardrooms and on Zoom calls to bicker over the color of the fire lane. They argue over which cloud provider to use, whether to adopt a new AI agent framework, or how to refactor a microservice—all while ignoring the structural fires burning through their runway. They copy-paste prior art, stack concrete band-aid on top of concrete band-aid, and then act astonished when their codebase calcifies into an unmaintainable, multi-million-dollar monument to technical debt.

I have spent over twenty years inside software systems, boardroom pitches, production incidents, hyperscaler infrastructure, and organizational scale-ups. From building native app engines over a weekend for ABC and Disney to architecting white-label media operating systems for Intel, to scaling enterprise AI retrieval infrastructure from pre-revenue to $14M ARR, my role has always been the same: Understand what is really going on.

> *“The ability to produce software implies a responsibility to change the world for the better.”*

This text is a rejection of technical theater. It is a first-principles blueprint for building software, products, and engineering organizations that are mathematically, architecturally, and culturally coherent from Day Zero.

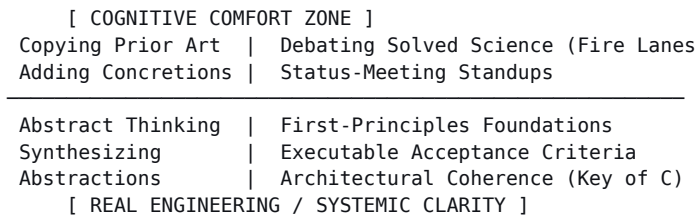
Chapter 1: Stop Painting the Fire Lane

In software, people love to talk about “bike-shedding”—complaining about the trivial while ignoring the essential. But bike-shedding implies the decision is purely subjective (like choosing what color to paint a bike shed).

Most of what happens in software organizations isn't bike-shedding. It's bickering over what color to paint the fire lane.

Fire lanes are red. The science is solved. The social norm is established. There is no legitimate reason on earth for a room full of highly paid adults to debate it. Yet 90% of the friction points inside engineering teams are about completely solved science—log capture mechanisms, deployment pipelines, test environments, or error modes.

It is the equivalent of a hospital board spending months debating handwashing protocols while patients are dying in the hallway.



The Delusion of Concretions

Why do smart people do this? Because dealing with concrete, familiar details feels safe to the lizard brain. Thinking in abstract, first-principles systems requires intellectual discipline.

Most software teams confuse familiarity with simplicity.

When confronted with a complex domain, an unprincipled team's instinct is to copy-paste prior art. They end up carrying ten thousand single-use screwdrivers in their toolbag because learning how a universal abstraction works feels “too abstract.” They complain loudly about “abstractions for abstraction's sake,” while simultaneously claiming that “the best PRs delete code.”

They miss the glaring contradiction in their logic: The only way to delete code without shrinking capabilities is to synthesize better abstractions.

If you build a system by stringing together thousands of concrete band-aids, you aren't building a platform. You're brewing primordial soup.

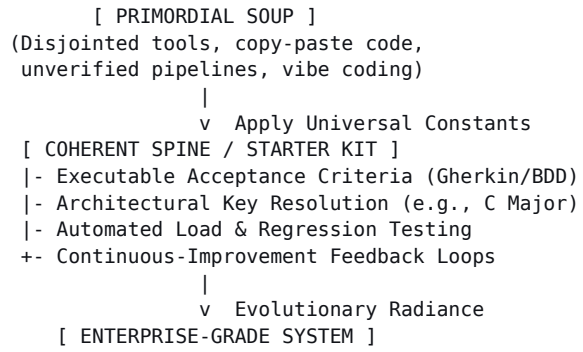
Chapter 2: Written in the Key of C — The Starter Kit

When life evolved on Earth, nature didn't reinvent genetics, cell membranes, or cellular respiration every time a new species emerged. It established a foundational architecture—a genetic code—and radiated outwards into millions of diverse organisms.

Yet in software, companies start from primordial soup every single time.

A seed-stage technology company gets funded, and instead of starting with a coherent spine, someone “vibe codes” a customer portal that silently redirects HTTPS back to HTTP because they lacked the conviction to enforce basic security protocols. They promise themselves they'll fix it later. But later never comes.

If I could pause time, I would build a universal Starter Kit for Technology Companies. Not a rigid methodology, but a set of non-negotiable universal constants that set the organization's key from Day Zero.



Setting the Key

Think of an engineering organization like a piece of music. If a company writes its entire codebase and culture in the key of C Major, and someone comes along and throws in an F-sharp because it felt convenient, it creates instant dissonance. It might not sound catastrophic to an untrained ear right away, but as the system grows, that discordance metastasizes into market underperformance.

Coherence is more important than local correctness or optimization.

Proof of Coherence: By the Numbers

Setting universal constants early isn't theoretical—it delivers order-of-magnitude business velocity:

- **Pre-Revenue to $14M ARR:** Evolving single-tenant proofs-of-concept into multi-tenant distributed architectures deployable in complex Fortune 100 environments.
- **6 Weeks down to 2 Days:** Reducing enterprise deployment cycles by 95% through automated installation patterns and custom PKI/mTLS security.
- **Minutes down to 3 Seconds:** Slashing runtime retrieval latency for mission-critical enterprise workflows.
- **$8M Annualized Savings:** Modernizing cloud-native architecture (e.g., Graviton EC2 migrations) across massive scale.

The Universal Constants of the Kit

- **Executable Specifications (Given / When / Then):** Hand over a folder of Gherkin files describing your product's behavior. Don't worry about how or when it runs. The system automatically verifies that the product matches the specification on every commit. If it doesn't, it doesn't ship.
- **Standardized Architecture Patterns:** Resolving core design and architectural patterns early so universal constants are set before edge cases perturb the system.
- **Automated Feedback & Load Profiling:** Continuous visibility into performance, latency, and failure modes before a single real customer experiences a slowdown.

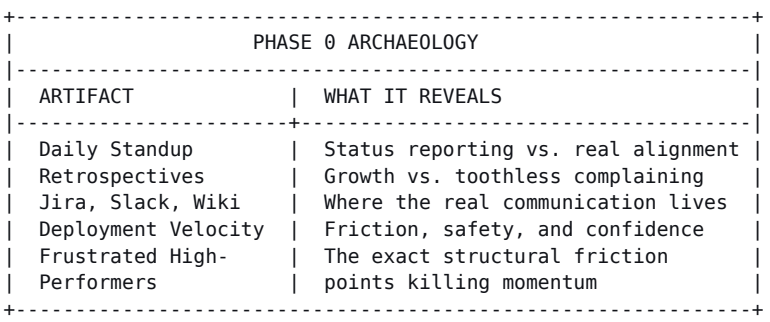
When you seed a company with these capabilities from the start, you don't spend Year Three retrofitting an exoskeleton onto a jellyfish. You start with a vertebrate.

Chapter 3: Archaeology — Finding Where the Meat Exists

When I walk into an organization in crisis, I don't start by reading their quarterly strategic slide deck. Slide decks are theater created for offsites. I conduct Phase 0 Archaeology.

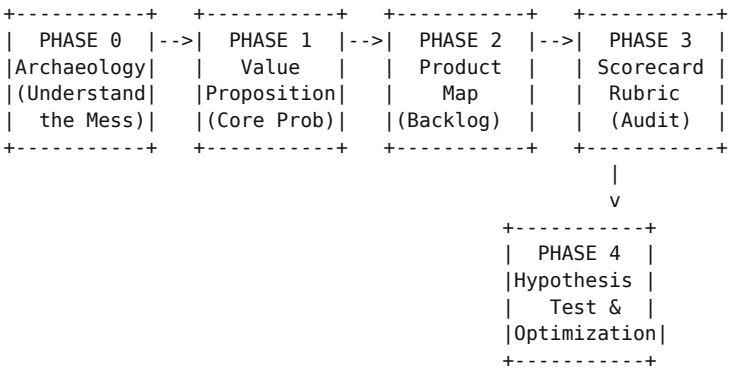
To understand what is really going on, inspect the actual artifacts:

- **Show me your daily standup:** Is it a 15-minute status meeting where developers report to a manager, or an operational alignment where the team unblocks work?
- **Show me your retrospectives:** Are they driving continuous improvement, or are they toothless complaints with zero action on the other side?
- **Show me your unplanned work:** Is it visible when developers are pulled off priorities to extinguish fires, or is it buried under the rug?
- **Show me your deployment pipeline:** If I ask today to change a comma to a semicolon in the product, how many hours or weeks before that change hits production?
- **Show me who is most frustrated:** Find the promising hire who is burning out. Find out why they are frustrated and who they are frustrated with. That is where the truth lives.



Chapter 4: The Four-Phase Execution Cadence

Transforming a software organization or building a platform from scratch requires a structured, 4-phase execution cadence:



Phase 0: Archaeology (Understand the Mess)

Interview the people closest to the work, map the communication paths, audit dev environments, and uncover the gaps between what leadership thinks is happening and what is happening.

Phase 1: Understand the Value Proposition

Isolate the customer's core problem. Not “What features can we build?” but “What problem does our customer have that software can solve?” Test assumptions against a small cohort of real users before writing code.

Phase 2: Map the Product & Clean-Slate Backlog

Use event storming, journey mapping, and user story generation to define the truly minimum viable solution. Decompose stories into small units backed by clear acceptance criteria. Legacy code doesn't get a free pass—it must earn its way into the new model.

Phase 3: The Scorecard Rubric

Evaluate both the software and the development process on an objective scorecard:

- **Software Scorecard:** Latency, throughput, failure modes, recoverability, observability, cost efficiency.
- **Process Scorecard:** Quality of specifications, acceptance criteria clarity, deployment speed, automated regression detection, product/engineering

negotiation balance.

Phase 4: Develop & Test Hypotheses

Formulate precise hypotheses for any score that falls short: “Adding database indexing here will reduce latency by 40%.” “Switching to Gherkin specs will eliminate sprint carry-over.” Execute, measure, iterate, repeat.

Chapter 5: First Principles in the Age of AI Data Planes

The explosion of AI models, Model Context Protocol (MCP) server architectures, and agentic coding workflows hasn't eliminated the need for principled engineering—it has removed every remaining excuse to be unprincipled.

If you feed an AI model or retrieval system messy, unstructured institutional knowledge full of copy-pasted concretions, the AI will signal-boost those bad patterns from its context window. It will generate thousands of lines of plausible-looking junk, compound your technical debt at lightspeed, and leave you with an unmaintainable system.

However, if you set your AI systems and agents up with first-principles context—unambiguous Given/When/Then acceptance criteria, clean MCP schema transforms, and strict architectural trust boundaries—AI becomes a massive force multiplier.

AI allows us to synthesize new abstractions and eliminate redundant code in seconds. But it requires human leaders operating from foundational truth to set the guardrails.

Epilogue: The Dawn of Things

There is a short window at the dawn of any technology company where the cost of staying committed to first principles is the lowest it will ever be—and the long-term benefit is the highest.

Yet leaders routinely surrender to short-term rushing. They convince themselves that cutting corners today is the only way to survive tomorrow, ignoring the reality that they are building a mountain of friction that will eventually bring them to a dead stop.

You do not have to choose between moving fast and building with discipline. When you start from first principles, you move faster, scale better, and hit fewer friction points from Day One.

It takes relentless energy. It takes a willingness to be misunderstood for long periods. It takes someone who looks at an impossible structural mess, finds clarity, and makes the system visible.

Let's get to work.

Let's get to work.

Chris Trahey

hello@christrahey.com · +1 (541) 390-9463