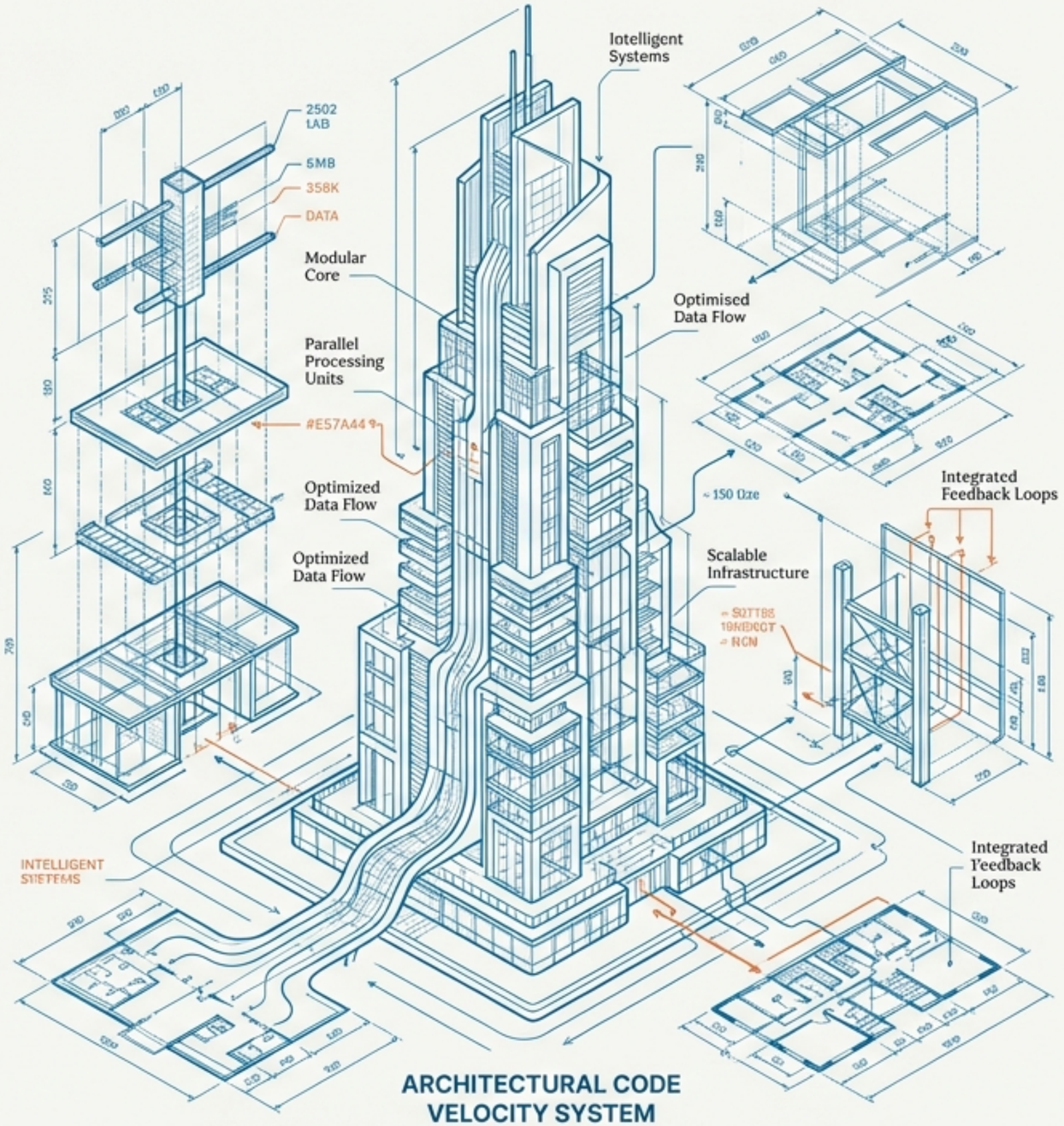
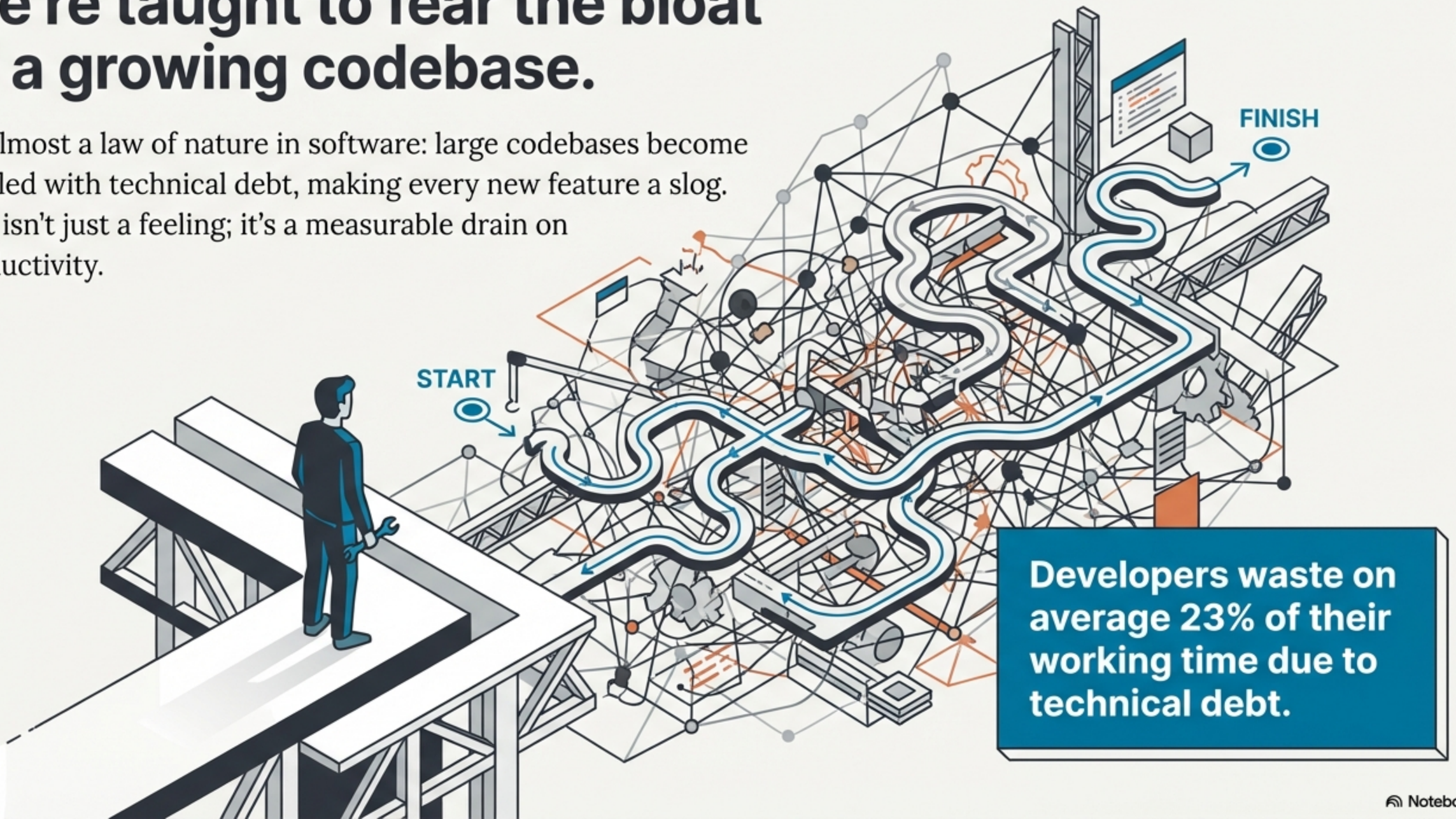
The Velocity Paradox

How More Code Can Make You Go Faster



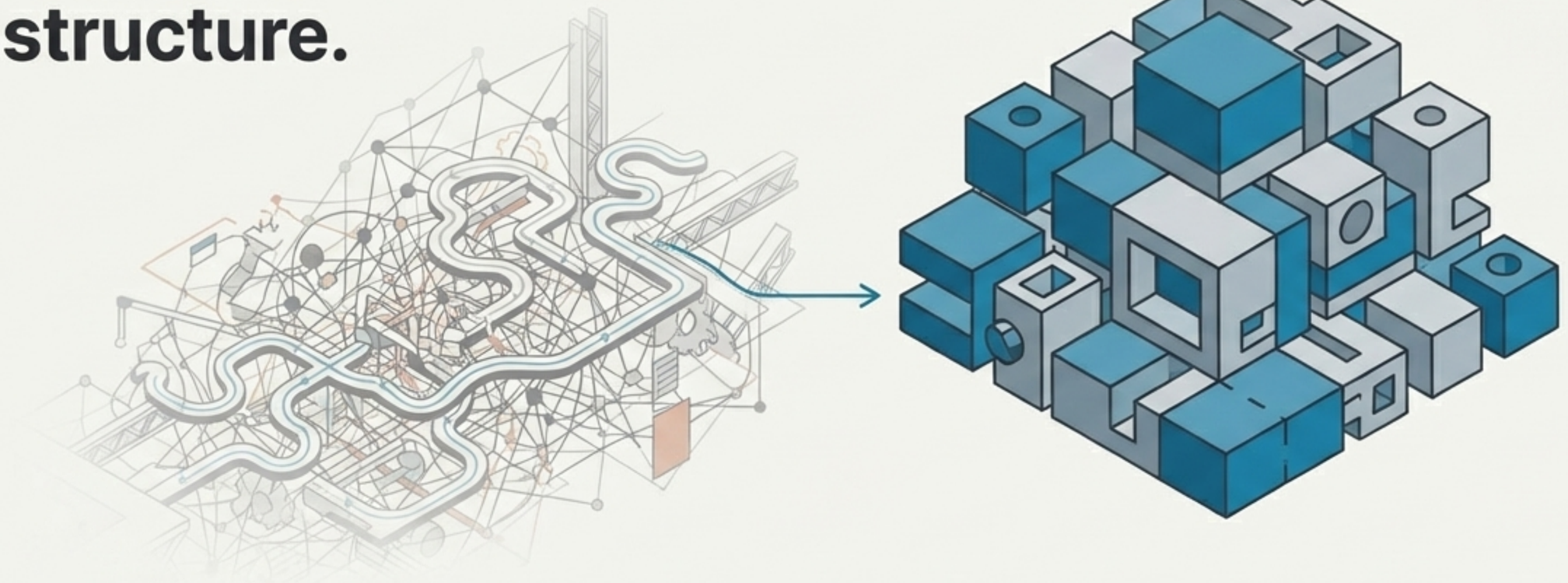
We’re taught to fear the bloat of a growing codebase.

It’s almost a law of nature in software: large codebases become tangled with technical debt, making every new feature a slog. This isn’t just a feeling; it’s a measurable drain on productivity.



Developers waste on average 23% of their working time due to technical debt.

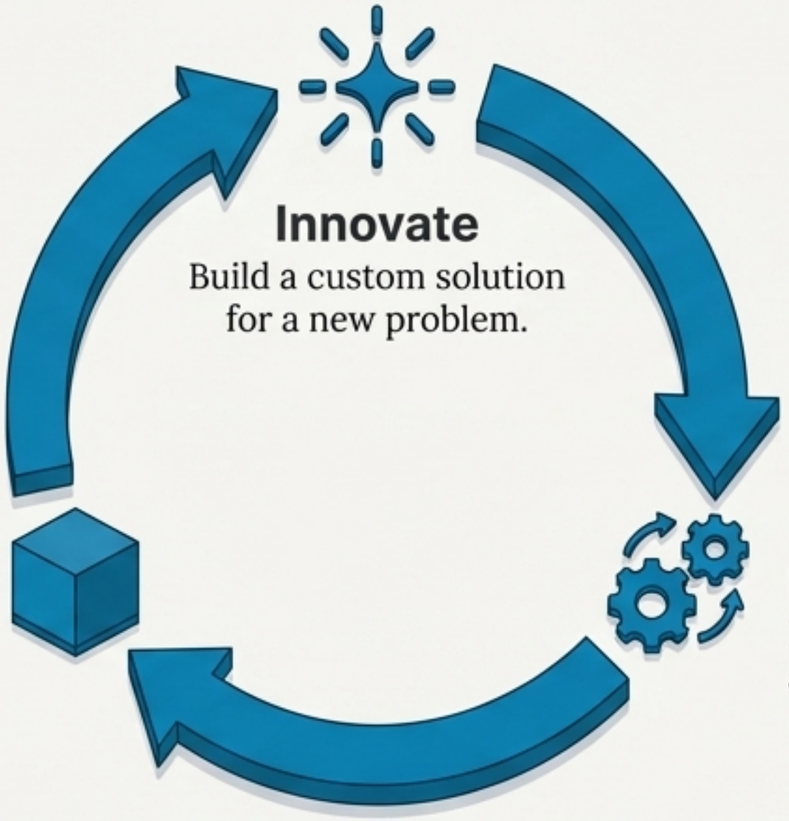
But the problem isn’t volume. It’s structure.



In a healthy system, code adds *capability*, not complexity. More capability means more speed.

A well-architected project with robust tests, clean abstractions, and smart reuse flips the script. Each additional component becomes an asset that accelerates development rather than slowing it down.

The Engine of Acceleration: The ILC Cycle



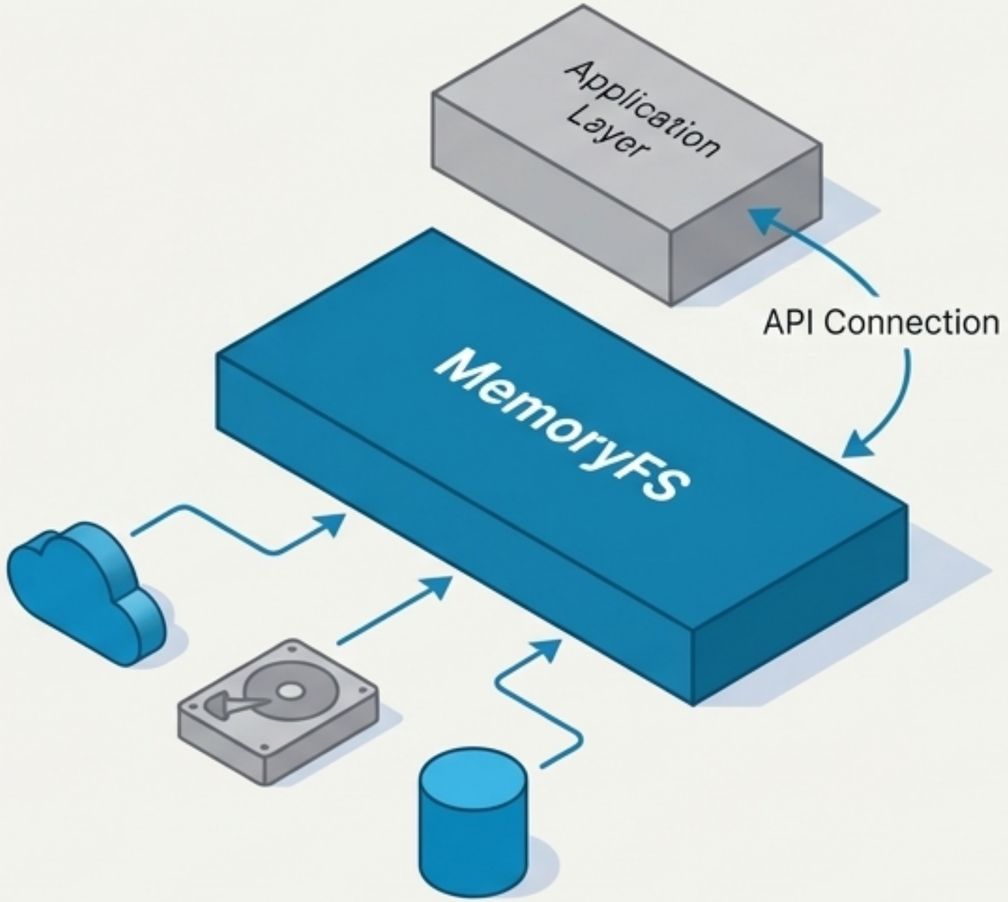
The Innovate, Leverage, Commoditize (ILC) pattern provides a strategic lens. We take a custom-built solution, refine it, and package it as a standardised, reusable component. As Simon Wardley says, you "turn it into a utility, allow everybody else to build on top of it... and that way, you just move up the stack."

Case Study: Building the Tower of Capability

Layer 1: Commoditizing Storage with an Abstraction Layer

We constantly wrote code to handle file storage across different environments: local disk, cloud storage, archives, databases.

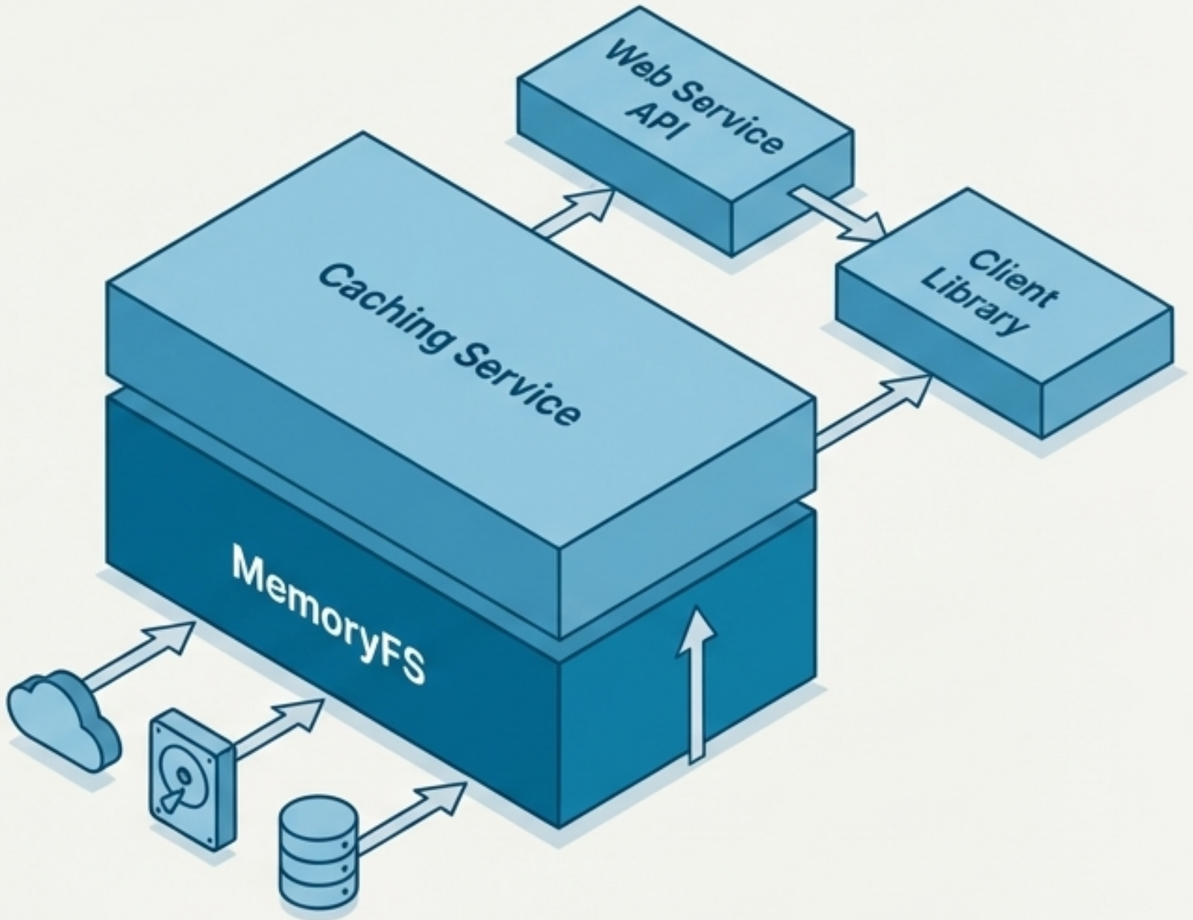
We built **MemoryFS**, an in-memory file system abstraction. This component allowed our code to save and retrieve files without caring where they were actually stored. It commoditized file storage into a single, self-contained module.



Layer 2: Building New Services on a Solid Foundation

On top of MemoryFS, we built a **Caching Service**. Because storage location was irrelevant, the caching service could introduce powerful strategies transparently. For instance, a single call can save a file to a “latest version” location and an archival folder simultaneously, giving us a current version and a historical backup automatically.

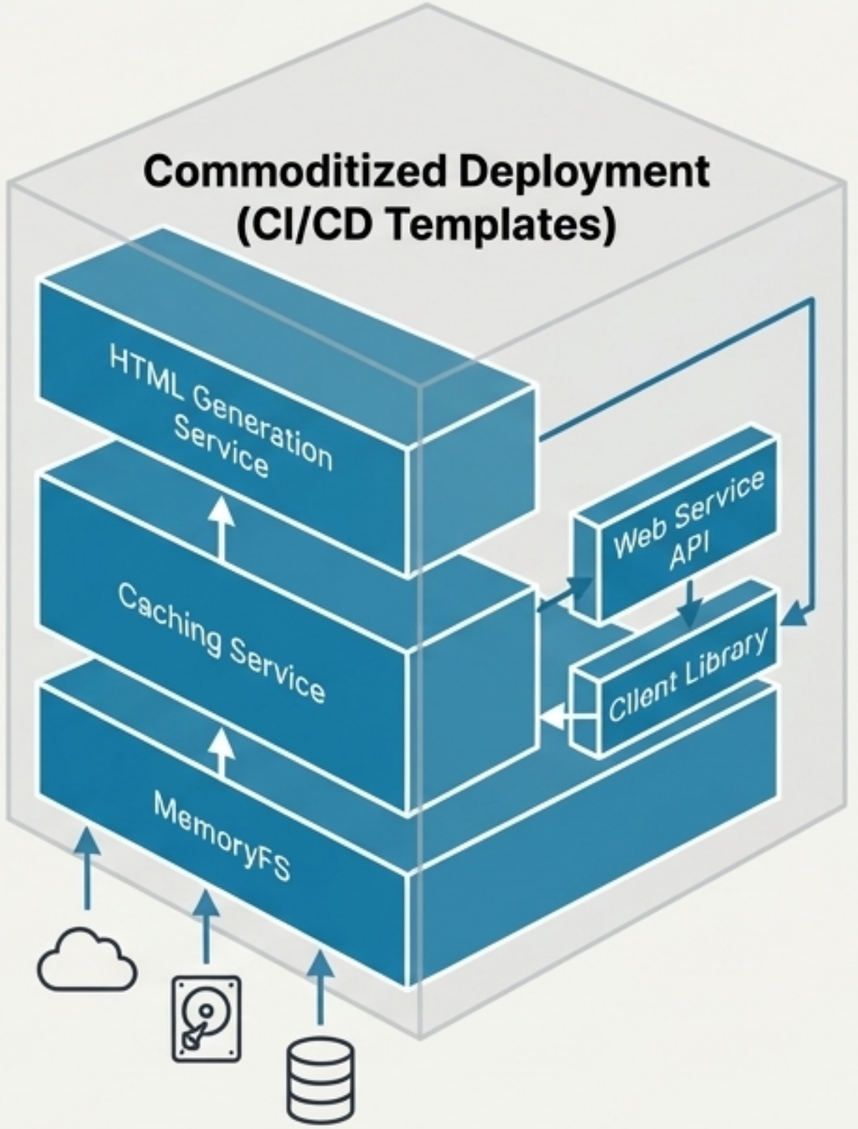
We then exposed this via a web service API and provided a lightweight client library, allowing any other tool to integrate with the caching service effortlessly.



The Full Stack: Assembling Systems, Not Just Writing Code

The layering continued. An **HTML Generation Service** used the caching client library directly, gaining caching capabilities without knowing any implementation details. Crucially, we also commoditized deployment itself. A base service template with CI/CD pipelines, testing, and deployment scripts baked in means new services can be deployed from day one without custom code.

Key Insight: We can now assemble new applications like snapping together LEGO bricks. Composition scales far better than construction, meaning our capacity to build accelerates as our library of components grows.



This creates a virtuous cycle of velocity.

Confidence & Stability

Building on a foundation of well-tested modules means less time debugging old code and more time writing new code.

Higher-Level Focus

Lower-level problems are solved once, freeing cognitive overhead. Each commoditized component enables a new genesis of ideas above it.

Parallel Development

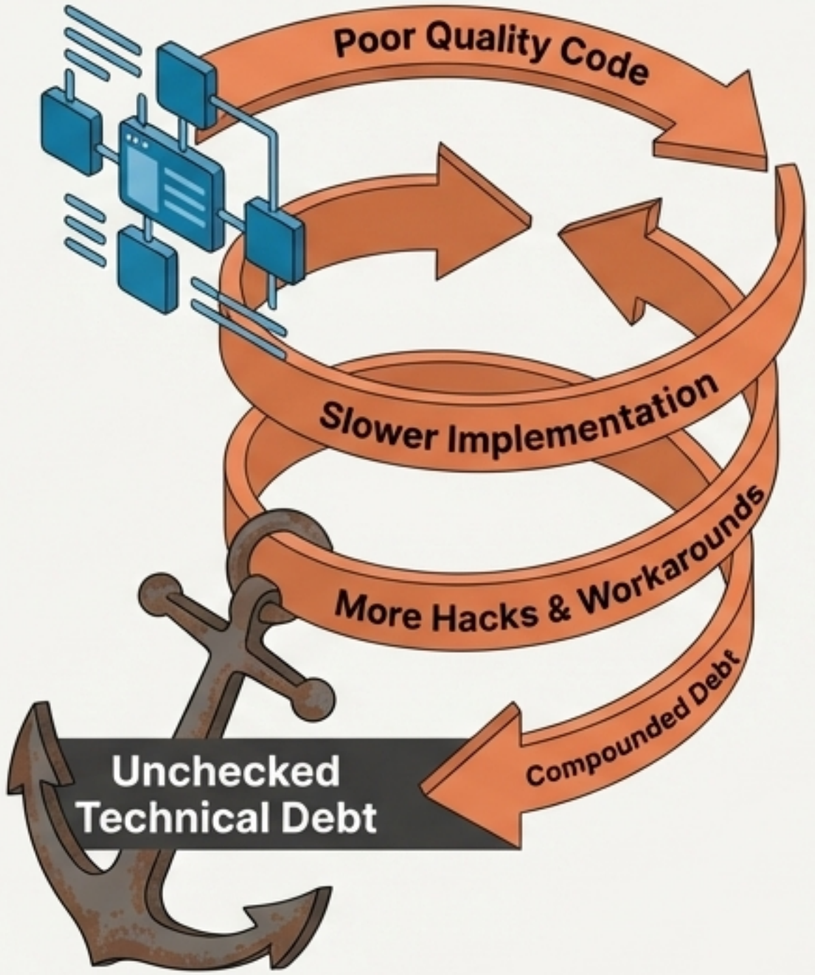
Clear module boundaries and standardised interfaces decouple development efforts, allowing teams to work independently.

Reduced Friction

Plug-and-play components and pre-built infrastructure streamline workflows, allowing developers to ship features faster and more reliably.

The alternative is a vicious cycle of debt.

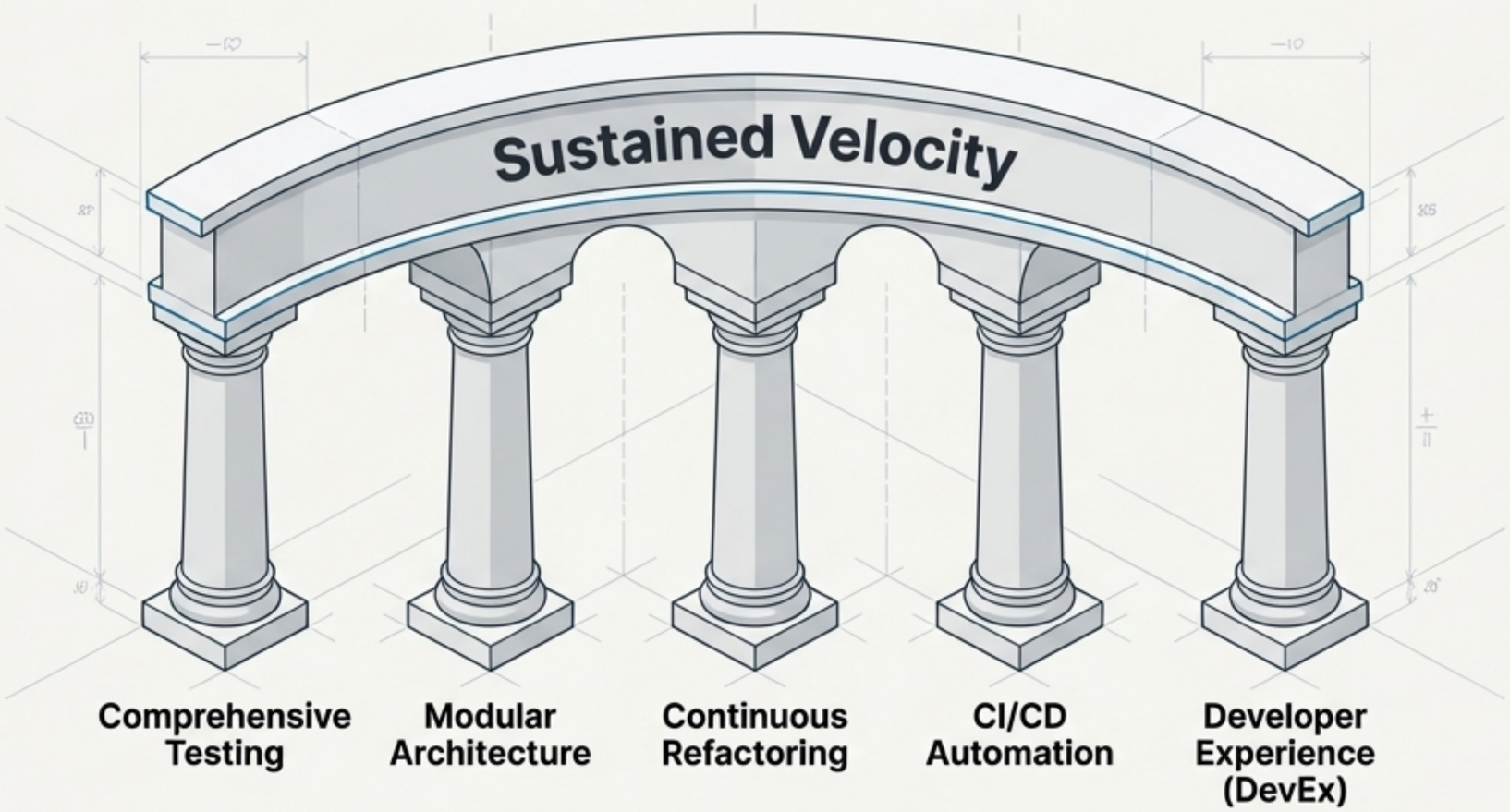
Unchecked technical debt—the quick-and-dirty solutions and poor-quality code—creates a downward spiral. Progress grinds to a halt as every every new feature requires wading through fragile components.

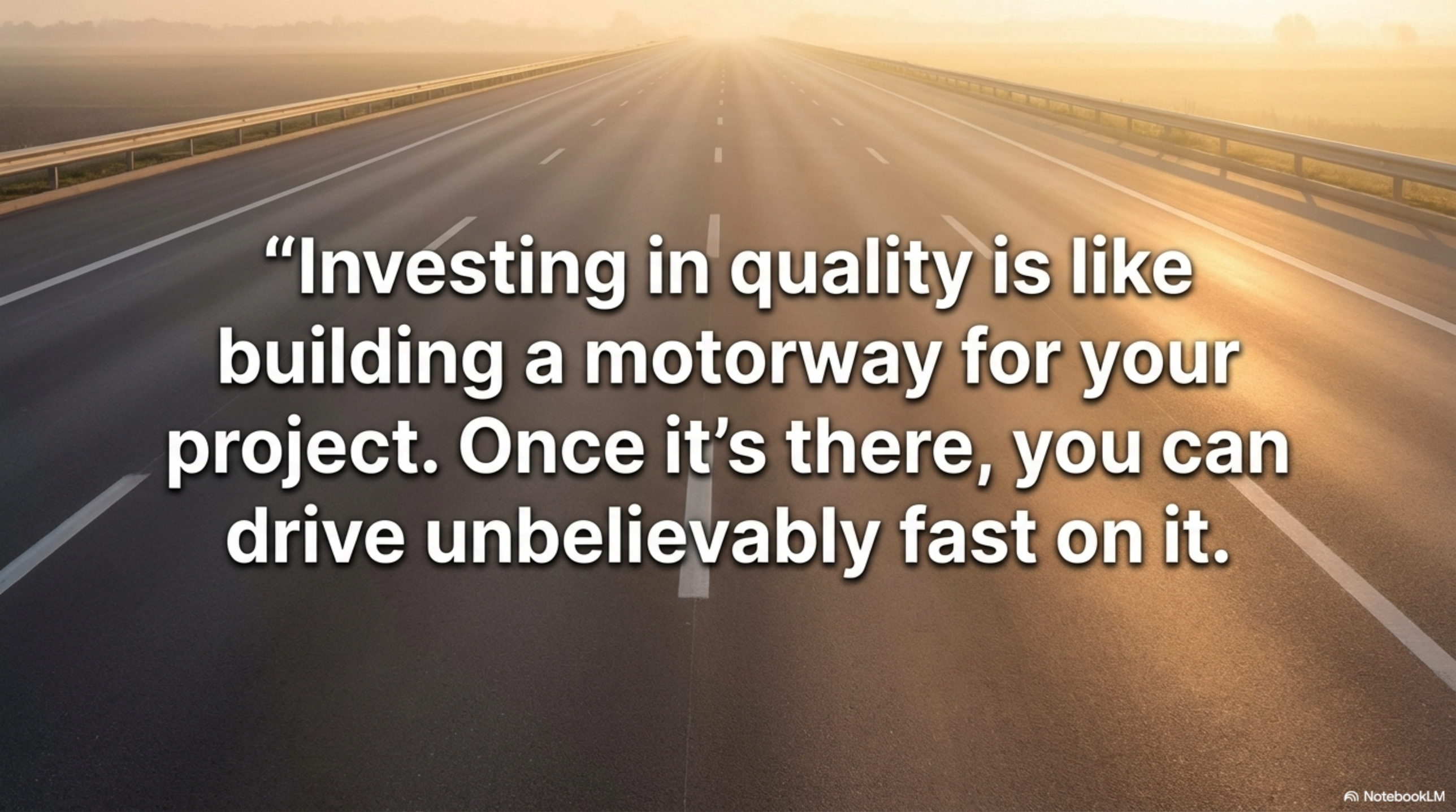


Nearly a quarter of the time when developers encounter existing debt, they are forced to introduce additional hacks or workarounds, compounding the problem.

To go fast, you must go well.

Sustained speed is built on a bedrock of engineering excellence. These non-functional requirements are the foundation of agility.

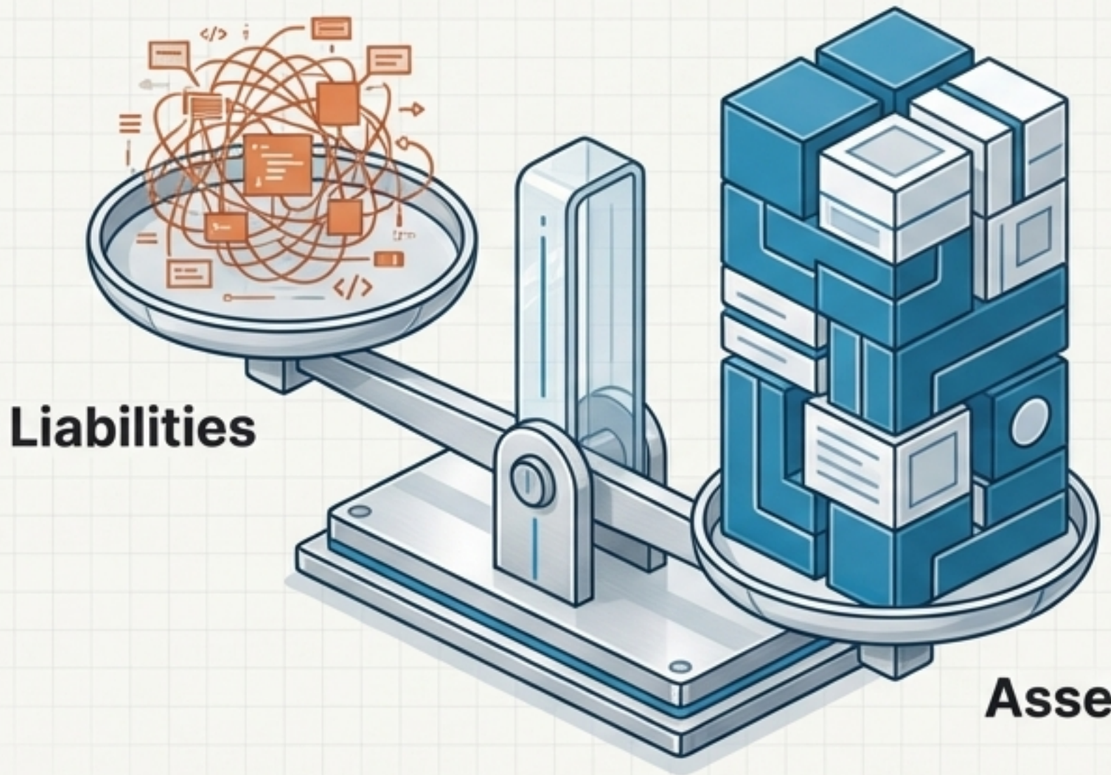



“Investing in quality is like building a motorway for your project. Once it’s there, you can drive unbelievably fast on it.

“Continuous attention to technical excellence and good design enhances agility.”

— The Agile Manifesto

It’s not about having more code. It’s about having more *assets*.



A well-tended codebase accrues assets, not liabilities. Each new module or service doesn’t add drag; it adds fuel to the engine. As the system evolves, it actually gains momentum.

Cultivate your codebase.

In a well-tended garden of code, growth leads to even more growth. **The more** quality code you cultivate, the faster you can harvest new innovation.