CI/CD for Business

Software developers have a term for the practice of continuously building, testing, and shipping small changes: CI/CD. Continuous integration, continuous deployment. You write a little code, merge it, test it automatically, and push it to production. The whole cycle might take minutes. It's one of the genuinely good ideas to come out of modern software engineering.

What's strange is that the companies shipping this software almost never apply the same discipline to themselves.

I've spent a lot of time evaluating technology companies — doing diligence, assessing how they build and operate. And there's a pattern I see constantly. The engineering team practices CI/CD religiously. They deploy dozens of times a day. They have automated test suites, feature flags, canary releases, the whole apparatus. Meanwhile, the business around them changes the way it works about once every eighteen months, usually badly, usually under duress, and usually by hiring a consulting firm to produce a PowerPoint deck that nobody follows.

The disconnect is remarkable. These companies have figured out that small, continuous, well-tested changes are the safest and fastest way to evolve software. But they treat organizational change as a big-bang event — something disruptive, risky, and best avoided until absolutely necessary. They've built a machine that can deploy code in minutes and strategy in years.

Why?

Part of it is that people confuse the word "innovation" with the act of innovation. Almost every company I encounter describes itself as innovative. They use the right vocabulary. Agility. Nimbleness. Continuous improvement. OKRs. But when you ask "innovative toward what end?" or "how, specifically?" the conversation gets vague fast. There's a gap between the language of innovation and the practice of it, and most companies live comfortably in that gap for years without noticing.

The deeper problem is that most companies have an incomplete conception of what innovation actually means. They think innovation is additive. You build new things. You launch new features. You enter new markets. And that's true, but it's only half the cycle.

In the 1950s, the economist Joseph Schumpeter described something he called creative destruction. His argument was that capitalism advances not just through invention but through the continuous replacement of old structures with new ones. The steam engine didn't just add a new capability to the economy. It destroyed the economic model built

around water wheels and horse-drawn transport. The replacement was the innovation, not just the invention.

Schumpeter's insight was that you can't have continuous innovation without continuous destruction. They're not separate processes. They're the same process. The new displaces the old, and that displacement is where the value comes from — not just because the new thing is better, but because removing the old thing frees up resources, attention, and organizational capacity that were being consumed by maintaining it.

Most companies get the first half. They're happy to build new things. It's the destruction part they can't stomach.

And you can see why. Destruction sounds bad. It sounds risky. It sounds like you're breaking things that work. The instinct is to preserve what exists while adding new capabilities on top. Keep the legacy system running. Maintain backward compatibility. Support the old API and the new API. Don't sunset anything because someone somewhere might still use it.

The result is accretion. Layer upon layer of systems, processes, and organizational structures accumulating over time, each one adding cost and complexity. The company gets heavier. Not in any single dramatic way — it's gradual, like gaining a pound a month. But after five years, you're carrying sixty pounds of organizational debt and wondering why you can't move as fast as you used to.

Does this sound familiar? It should. It's the same problem software engineers face when they don't refactor. When you keep adding code without removing or restructuring what's there, you get a codebase that's increasingly expensive to maintain and increasingly dangerous to change. Every software engineer knows this. The solution is continuous refactoring — small, ongoing improvements to the structure of the code that keep it healthy and adaptable.

The business equivalent of refactoring is creative destruction. But while software teams refactor routinely, business teams almost never do. They wait until the pain is unbearable, then launch a massive transformation initiative that takes two years and costs millions. This is the organizational equivalent of a complete rewrite — the thing every experienced developer knows is almost always the wrong approach.

What would it look like to apply CI/CD thinking to the business itself?

It would mean treating organizational change as normal, not exceptional. Not something you do once a year in a strategic planning offsite, but something you do continuously, in small increments, with feedback loops that tell you what's working and what isn't.

It would mean retiring things. Actively, deliberately sunsetting old processes, old tools, old ways of working that no longer serve the mission. Not in a big dramatic purge, but continuously — the same way a good engineering team continuously removes dead code and deprecated APIs. Every time you add a new capability, you ask: what does this replace? And then you actually replace it.

It would mean measuring what matters. Not just activity metrics — how many features shipped, how many meetings held, how many initiatives launched — but outcome metrics that tell you whether ways of working are actually improving. Are decisions faster? Are customers better served? Are people spending less time on maintenance and more on creation? Software teams have DORA metrics for this. Most business teams have nothing comparable.

It would mean building for change from the beginning. One of the things that kills companies is that their products and processes are designed as if the current state is permanent. They optimize for today at the expense of tomorrow. A CI/CD mindset means building with extensibility in mind — not because you know exactly what will change, but because you know something will.

Here's what I think companies get most wrong about agile methodology, and it connects directly to this. Agile was supposed to be about responding to change. But in practice, most agile implementations have become process religion — obsessed with ceremonies, sprint rituals, story point estimation, and methodology compliance. The focus has shifted from what you're building and why to how everyone is working and whether they're following the rules. The backlog becomes a queue of small tasks disconnected from any larger vision. Iteration becomes an end in itself rather than a means to an end.

The original agile insight was sound: you can't plan everything upfront, so build iteratively and adapt. But somewhere along the way, the "adapt" part got lost. Teams iterate without ever stepping back to ask whether they're iterating toward the right destination. They deliver continuously without destroying continuously. They add without subtracting. The result is software that technically ships frequently but strategically drifts.

This matters enormously because of what technology now makes possible. We're living through a period where the cost of building new capabilities is dropping dramatically. Cloud services let you stand up infrastructure in hours that used to take months. LLMs let a small team do work that used to require a department. Automation tools handle repetitive processes that used to consume entire roles. The potential to work differently — not just faster at the same things, but fundamentally differently — has never been greater.

But you can't capture that potential if you're spending all your energy maintaining the old. Every legacy process you preserve, every deprecated system you keep alive, every organizational structure you maintain because changing it would be uncomfortable — these are direct taxes on your ability to move forward. The companies that will win aren't the ones that add new technology the fastest. They're the ones that retire old ways of working the fastest.

I think this is why creative destruction feels so counterintuitive to most leaders. We're wired to view loss as worse than an equivalent gain. Giving something up — even something that's clearly holding you back — feels risky in a way that adding something new does not. So companies keep accumulating. They build the new chatbot without retiring the old phone tree. They adopt the new project management tool without sunsetting the old one. They hire for new capabilities without reorganizing around them. Every addition comes without a corresponding subtraction, and the organization slowly calcifies.

The fix isn't a better strategic planning process. It isn't a digital transformation initiative. It's a change in disposition. It's deciding that organizational change is a continuous process, not a periodic event. It's applying to the business the same principle that makes CI/CD work in software: small changes, shipped frequently, tested against reality, with the discipline to remove what no longer works.

Software's CI/CD works because it treats change as the default state, not the exception. The pipeline is always running. The tests are always checking. The deployments are always flowing. Nothing is permanent. Everything is subject to the next commit.

Imagine running a company that way. Not recklessly — CI/CD isn't reckless, it's disciplined. But with the assumption that what you're doing today will be different from what you're doing in six months, and that the transition between the two should be smooth, continuous, and unsurprising. Where retiring the old is as celebrated as launching the new. Where the cost of not changing is measured as carefully as the cost of changing.

Schumpeter understood that the cycle of creation and destruction isn't a bug in capitalism. It's the mechanism. The companies that grasp this — that build the destruction pipeline alongside the creation pipeline — are the ones that stay fresh, that stay fast, that don't wake up one morning buried under a decade of accumulated ways-we've-always-done-it.

The examined business, like the examined codebase, is the only one worth shipping.