

BEYOND THE GREEN BUILD

The Last Mile to Production: Why Continuous Delivery Is a Concept Long Before It Is a Reality

On the distance between a build that is ready on every commit and an organisation that still ships once a quarter — and the structural forces that keep them apart



Giovanni Leonardi

November 2010

Contents

1. Executive Summary	3
2. The Team That Could Ship, and Didn't	3
3. A Promise, Newly Named	4
4. Where the Promise Meets the Building	4
5. The Forces That Hold the Gap Open	5
5.1 The organisation is split down its middle	5
5.2 Control is achieved by inspection	5
5.3 The path to production is nobody's product	5
5.4 Environments are scarce, shared and unlike the real thing	6
5.5 The release is a ritual, and rituals resist their own abolition	6
6. In Defence of the Wall	6
7. Closing the Distance Without Pretending It Is Easy	7
8. The Gap as a Mirror	7

The Last Mile to Production: Why Continuous Delivery Is a Concept Long Before It Is a Reality

Programme · No. 156

Originally written November 2010 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

“The last mile stays manual because manual is free until the weekend it isn't.”

1. Executive Summary

Somewhere in most large delivery organisations today there is a team that can, in the narrow technical sense, release its software at almost any moment. Every change its developers make is compiled, tested and assembled into a working build within minutes. On the wall there is a dashboard, and the dashboard is a reassuring green. And yet that same software reaches the people who use it once a quarter, through a manual, rehearsed, all-hands weekend held together by a printed checklist and a few people who have done it enough times to be trusted with it.

This essay is about the distance between those two facts, because that distance is where most of the truth about a delivery organisation lives. A new idea has been given a name this year – continuous delivery, the practice of keeping software permanently ready to release and moving each change to production in small, frequent, low-drama increments. It is a good idea, and it is spreading fast as an aspiration. What is spreading far more slowly is the reality.

The argument I want to make is that the gap between the concept and the practice is not, at bottom, a technical problem, and will not yield to a technical answer alone. It is held open by structural forces: an organisation split between those who build and those who run; a model of control that works by inspecting change rather than building quality in; a path to production that no one is funded to own; environments that are scarce, shared and unlike the real thing; and a culture that treats the release as an event to be survived rather than a routine to be forgotten. None of these is foolish. Together they are why continuous delivery is so easy to believe in and so hard to live. Closing the gap is less a matter of adopting a tool than of rebuilding how an organisation earns its own confidence – and that is slow work.

2. The Team That Could Ship, and Didn't

Consider a team I will keep deliberately composite, because the shape of it is everywhere. It had done, by the standards of 2010, almost everything right. Developers integrated their work continuously; a build server assembled the whole system on every change and ran a growing suite of automated tests; a green build meant, genuinely, that the software worked. From a standing start the team could produce a deployable package in about nine minutes.

The lead time from a developer finishing a change to that change being used by a customer was eleven weeks.

The nine minutes and the eleven weeks lived in the same organisation, and no one experienced them as a contradiction, because they belonged to different worlds. The nine minutes belonged to development, where automation was prized and improvement was continuous. The eleven weeks belonged to everything after: a queue for one of three shared test environments, booked weeks ahead and never quite matching production; a hand-off to a separate operations group by way of a written request; a change advisory board that met on Thursdays to approve what would go where; and, at the end, a release — a single scheduled weekend a quarter, some fifty manual steps long, during which the whole apparatus held its breath.

Ask anyone on that team whether they did continuous delivery and they would have said yes, and pointed at the green dashboard. They were not lying. They had built the first, easy, visible half of it. The second half — the part that actually delivers — ran straight into the structure of the organisation and stopped.

3. A Promise, Newly Named

It is worth being precise about what the promise is, because the phrase is new enough that people mean different things by it. Continuous delivery is not simply continuous integration done a little more often, though it grows from that root. Continuous integration solved the problem of developers' work drifting apart by merging and building it constantly; continuous delivery extends the same instinct all the way to the user, insisting that the software be kept, at every moment, in a state from which it could be released. Releasing becomes a business decision taken whenever it suits, not a technical ordeal scheduled around the calendar.

The mechanism it proposes is a pipeline: a change moves through a sequence of automated stages — build, automated test, and on into progressively more production-like environments — and any stage that fails stops the change cold. What survives to the end is, by construction, a release candidate. The lineage of the idea is easy to trace. It takes from the agile movement the habit of small increments; it takes from lean thinking, much in the air just now, the conviction that small batches and smooth flow beat large batches and big bangs; and it draws on a newer, rougher current — a coinage, devops, that has begun to circulate at a handful of conferences, naming the suspicion that the wall between development and operations is the thing most in need of demolition.

Strip away the machinery and the promise is really about a state of being rather than a set of tools: an organisation that is always ready, so that release is boring. Boring is the aspiration. It is the least glamorous and most radical thing the idea asks for.

4. Where the Promise Meets the Building

The pipeline is real, and where it exists it works, right up to the point where it meets the boundary between the people who build the software and the people who run it. There the automation tends to stop, and the tickets begin.

The concept	The common reality
Every change is a potential release	Changes accumulate for a quarter, then release together
Releasing is a business decision	Releasing is a scheduled technical event

The concept	The common reality
The pipeline runs to production	The pipeline runs to a hand-off, then people take over
Environments are consistent and disposable	Environments are scarce, shared and subtly unlike production
Control is evidence the pipeline produces	Control is approval granted in a meeting
Done means released	Done means code-complete and passed to someone else

Each row of that table is a place where an elegant idea meets an organisation and loses. The change that could be a release becomes one line item in a quarterly batch. The environments that should be produced on demand are instead a booking problem. The confidence that the pipeline could generate as evidence is generated instead by a person signing a form. The gap is not a gap in understanding — the teams grasp the idea perfectly well — but a gap between the idea and the arrangements it has to survive.

5. The Forces That Hold the Gap Open

If the gap were mere ignorance it would already be closing, because the concept is neither secret nor hard to grasp. It persists because a set of structural forces actively hold it open, and each of them made sense on the day it was put in place.

5.1 The organisation is split down its middle

The oldest and deepest force is the division between development and operations. They report through different lines, answer to different managers, and — this is the crux — are rewarded for opposite things. Development is paid to change the system; operations is paid to keep it stable. A release is the moment those two mandates collide, and the organisation has resolved the collision by building a wall and passing work across it through a slot. Continuous delivery asks the wall to come down. Nothing in either group's incentives rewards them for taking it down, and a good deal punishes whoever is on shift when the resulting change breaks.

5.2 Control is achieved by inspection

Large organisations have learned, often through expensive failures, to control change by inspecting it. A board reviews what is proposed; an approval is granted or withheld; a record is kept. In regulated sectors this is not optional, and even where it is not mandated it is deeply reasonable: someone senior looks before the organisation leaps. The trouble is that inspection scales badly. It assumes changes are large and infrequent enough to be examined one at a time by people in a room. Continuous delivery makes changes small and frequent, which is precisely the shape that inspection-by-committee cannot process. So the pipeline fills up behind the meeting, and the batch re-forms, and the quarter reasserts itself.

5.3 The path to production is nobody's product

Ask who is funded to own the route from a finished change to a running feature and, in most organisations, the answer is no one. Projects are funded to build things. The pipeline, the deployment automation, the environment provisioning — the whole apparatus that

turns built into live — is overhead, done in the margins, first to be cut when a deadline looms. Done is defined as code-complete, which quietly declares the last mile to be someone else's problem. So the automation that development lavishes on the build peters out at exactly the point where it would do the most good, and the last mile stays manual because manual is free until the weekend it isn't.

5.4 Environments are scarce, shared and unlike the real thing

The physical facts resist the idea too. Production is a place with real data, real scale and real integrations; the environments that stand in for it are few, shared between teams, and never quite the same. Virtualisation is starting to ease this, but slowly, and the enterprise reality of 2010 is still a queue for a handful of staging systems that drift out of alignment the moment they are configured by hand. A pipeline is only as trustworthy as the environments it runs through. When those environments are scarce and inconsistent, every promotion is a small gamble, and the organisation rationally responds by promoting rarely and watching closely.

5.5 The release is a ritual, and rituals resist their own abolition

Finally, and least discussed, there is temperament. The quarterly release weekend is difficult, but it is also a known quantity, and around it has grown a certain pride. People know their part. There is a war room, a bridge call, a sense of shared endeavour and, on Monday, relief. Continuous delivery proposes to make all of that disappear — to render the release so unremarkable that no one notices it. That is the goal, and it is also, quietly, a loss: of ritual, of visible heroism, of the reassuring feeling that something important is being handled carefully because it is being handled slowly.

The manual release survives not because people believe it is good, but because everything around it — how the organisation is split, funded, governed and reassured — is arranged as though it were necessary. Change the release without changing its surroundings, and the surroundings win.

6. In Defence of the Wall

It would be easy, and it is becoming fashionable, to treat all of this as simple backwardness — the enemies of progress guarding their meetings. That is too cheap, and getting it wrong matters, because the case for the current arrangement is stronger than its critics allow.

The wall between building and running encodes real experience. Production failures are expensive, sometimes dangerous, and in some industries a matter of law; the people who insist on caution are usually the ones who have been called out at three in the morning to clean up an enthusiast's mistake. Change control exists because unreviewed change has, historically, hurt people. The separation of duties that so frustrates the pipeline is, in some sectors, a control an auditor will require by name. When operations resists just ship it, they are frequently right about their own context in a way the developer proposing it is not.

The honest version of the continuous-delivery argument therefore cannot be that control is unnecessary. It is that the current method of control — inspection after the fact, by people, in batches — is not the only way to achieve it, and is increasingly a poor one. The

question is not whether to have control but where to put it: at the end, as a gate, or throughout, as a property of the way the work is done.

*“The opposite of a risky release is not a carefully inspected one.
It is a boring one.”*

7. Closing the Distance Without Pretending It Is Easy

If the gap is structural, then only structural moves will close it, and none of them is quick. From where we stand in 2010, with the tools still immature and almost no one able to claim they have finished the journey, the direction is nonetheless becoming visible.

1. Bring the running of the system into the building of it. The wall is the first thing to go. That does not mean abolishing operations; it means making the people who run the system responsible participants in how it is built and released, and making developers live with the consequences of what they ship. Shared responsibility is the precondition for everything else.
2. Fund the path to production as a product. The pipeline, the deployment automation, the provisioning of environments — treat these as first-class deliverables with owners and budget, not as overhead scavenged from project margins. What is not funded will not be built, and the last mile is not funded.
3. Make environments and configuration something you produce, not something you possess. The scarcity that forces the queue eases only when a production-like environment can be created on demand and configured by script rather than by hand. The means to do this are young and rough today, but the intent is clear: an environment should be an output, not an asset.
4. Redefine done as released. As long as completion means code-complete, the gap has somewhere to hide. Move the definition to the far end — a change is done when it is safely in the hands of users — and the whole organisation is suddenly interested in the last mile it used to ignore.
5. Let the pipeline generate the evidence the meeting used to provide. This is the hardest and most important. Control need not be surrendered to gain speed; it can be rebuilt so that the automated pipeline itself produces the record, the tests, the traceability and the reversibility an approver was signing for. Governance stops being a gate the work waits behind and becomes a property the work carries with it.

None of this is a weekend's work, and anyone who sells it as a product to be installed has misunderstood it. It is organisational redesign wearing the costume of a technical practice, and it will take years and cost political capital, because it asks groups to merge, budgets to move, and senior people to trade the comfort of the gate for the discipline of the pipeline.

8. The Gap as a Mirror

What makes continuous delivery worth thinking about, beyond its considerable practical value, is what it reveals. The technical half of it — the build, the tests, the pipeline — turns out to be the easy half, and a determined team can reach it in months. The moment the

pipeline reaches the wall, it stops being a story about software and becomes a story about the organisation: how it is divided, what it rewards, whom it trusts, and what it is afraid of.

The distance between a build that is ready on every commit and an organisation that ships once a quarter is not really a distance in engineering. It is the same distance that opens up in every transformation between what we sincerely intend and what our structures will actually permit. Continuous delivery is a mirror held up to the delivery organisation, and most of us, looking into it just now, are seeing the green dashboard and the printed checklist side by side, and only beginning to admit that the second is the truer picture.