

TECH & AI CASE STUDY - ENTERPRISE SOFTWARE - 2025

CI/CD Modernization: From Quarterly Releases to Daily Deploys

Based on a delivery engagement with Copperline Software

quarterly to daily
RELEASE CADENCE

**~90 days to 2
days**
LEAD TIME TO PRODUCTION

-60%
CHANGE FAILURE RATE

days to <1 hr
MTTR

A fillable practice edition: the full story, 8 actions,
6 reflections and a 90-day plan - work it with your team.

Free - no email required
emersontechnology.com

MAKE THIS PRACTICE BOOK YOURS

Work it, do not just read it

This practice book turns the Copperline Software case study into a working session for your team. Read the full story first - what happened, what was built and what it changed. Then tick the 8 actions you can commit to, write short answers to the 6 reflections, and finish with a 90-day plan. Every field in this PDF is fillable: open it in Adobe Acrobat Reader, Chrome, Edge or macOS Preview, type directly into the boxes and save.

YOUR NAME

ROLE

COMPANY

TEAM / DEPARTMENT

DATE

EMAIL (OPTIONAL)

MY GOAL FOR THIS PRACTICE SESSION

- 01

The story - What happened at Copperline Software - before, during and after the project.
- 02

The results - The numbers that changed, plus industry benchmarks from public research.
- 03

The 8-action checklist - Tick boxes as your organization completes each step. Owner and date fields included.
- 04

The 6 reflections - Fillable questions that pressure-test your own roadmap.
- 05

My 90-day plan - Three fillable phase cards that turn this case study into your roadmap.

RUN IT AS A 90-MINUTE TEAM SESSION

25 min READ Read the story together. Ask everyone to note the one number that surprised them most.	40 min PRACTICE Work the 8-action checklist and the 6 reflections. Assign an owner and a date to every action you commit to.	25 min COMMIT Fill in the 90-day plan. One goal per phase, one number that tells you it worked. Book a review date before you leave the room.
--	--	---

THE STORY

What happened, and why it worked

01 The Company

Copperline Software is a 430-employee vendor of an ERP suite used by 400 mid-market enterprises across manufacturing, distribution and professional services. Its customers run finance, inventory and operations on the platform, which makes Copperline's release cadence a strange kind of public commitment: four times a year, in release trains, customers receive a bundle of everything the company has built, whether or not they asked for it. The engineering organization is large enough to have real specialization — product teams, a QA group, a release-management function — and sophisticated enough that customers integrate with its APIs, run custom workflows and depend on it for quarter-end and year-end closings when downtime is commercially intolerable. Copperline's market was moving faster than its delivery model. Competitors built on modern delivery pipelines were shipping weekly, and prospective customers had begun asking pointed questions in evaluations: how long does a requested change take to reach production, how often do you deploy, what happens when a release goes wrong. Copperline's sales team could not answer those questions with numbers they were proud of, and the gap between the product's depth and its delivery mechanics had become a visible commercial weakness.

02 Where Things Stood

Every quarter began the same way: a six-week code freeze while a manual QA cycle ground through the accumulated quarter's work. Long-lived release branches made merges genuinely hazardous — teams would spend days reconciling divergent code, and the merge conflicts themselves introduced defects that QA then had to find again. Once QA passed, the release weekend arrived: 60 engineers coordinated manual deployment steps across customer environments, each installation with its own configuration drift, running through checklists that only a handful of veterans fully understood. Hotfixes piled up between trains, so the next release carried an even heavier payload, which made the next freeze longer and the next release riskier — a feedback loop that tightened quarter after quarter.

The human and commercial costs were intertwined. Customers routinely waited two quarters for changes they needed in weeks, and account managers had learned to apologize for it. The pattern broke into the open with a botched October release that forced a weekend-long rollback across 40 customer installations, consuming an estimated 600 engineering hours and pulling account teams into damage control with some of the company's largest accounts. Leadership could see the pattern clearly by then: release weekends mobilized a small army, change failure was common enough to plan around, and competitors shipping weekly were winning deals on velocity alone. The delivery process was not merely slow — it was the company's largest operational risk.

03 The Challenge

The challenge was modernizing delivery for an installed-base product without breaking the trust of 400 enterprises who had built processes around quarterly predictability. Google's DORA research consistently reports that elite performers deploy on demand, multiple times per day, with change lead times measured in hours rather than months — but Copperline's entire organizational choreography, from QA staffing to release management to customer communications, was built around the opposite assumption. Changing cadence meant changing contracts, expectations and habits simultaneously, not just swapping tools.

There were three specific structural problems. First, the branch model manufactured risk: long-lived release branches guaranteed that integration pain arrived all at once, at the worst possible time. Second, the QA model could not scale: manual regression testing a quarter's worth of changes took six weeks, so faster cycles were mathematically impossible without automation. Third, deployment was artisanal: every customer environment was

a snowflake, and Puppet's State of DevOps research found high-performing IT organizations deploying code 46 times more frequently than low performers, with faster lead times and lower failure rates — a gap Copperline could not close while deployments remained hand-crafted events requiring 60 people. Any solution had to address all three, in an order that kept production stable while the machinery was rebuilt underneath it.

04 Discovery and Diagnosis

Emerson Technology insisted on a measurement quarter before any tooling changed. The team recorded deployment frequency, lead time, change failure rate and MTTR across one full release cycle, interviewing engineers, QA analysts, release managers and account managers as the data accumulated. The baseline confirmed what everyone suspected and quantified what nobody could prove: lead time from merge to production averaged roughly 90 days, deployments were quarterly events, and the change failure rate was high enough that release weekends carried a standing rollback plan. Just as valuable, the interviews located the knowledge concentration — the deployment checklists lived in the heads of a few release veterans, a bus factor of essentially two.

The diagnosis also mapped where failure actually originated. Contrary to the assumption that new features were the danger, the change-failure analysis showed that most production incidents traced back to merge conflicts and configuration drift between customer environments — risks created by the process itself rather than by careless engineering. Database schema changes emerged as a second chronic blocker: because migrations required coordinated downtime, schema work was batched into release weekends, which made every release larger. The diagnosis ended with a three-stage roadmap — branch model first, test automation second, progressive deployment third — sequenced so that each stage reduced the risk of the stage after it, and with explicit commitments that customer-facing stability would be measured weekly throughout the transition.

05 The Solution

The modernization ran in three deliberate stages, each dismantling one structural problem while the product continued shipping to customers every quarter.

Trunk-based development with flags

The codebase moved to trunk-based development with feature flags, ending the six-week code freezes entirely. Engineers merge small changes to trunk continuously; incomplete work ships dark, hidden behind flags that activate per customer. Long-lived release branches — where merge pain and release fear were manufactured — disappeared, and with them the quarterly merge crisis. Flags also created a commercial capability the company had never had: features can be enabled for a pilot group of customers, validated in production, and rolled out progressively, which turned every release from a bet into a controlled experiment.

The rebuilt test pyramid

The monolithic manual QA cycle was replaced by a deliberately reconstructed test pyramid: a fast unit suite, contract tests between services, an expanded integration suite and a thin end-to-end layer, all running in GitHub Actions with parallelized jobs completing in under 20 minutes. Every pull request spins up an ephemeral preview environment with realistic seeded data, so reviewers test actual behavior instead of imagining it from diffs, and QA engineers stopped being a bottleneck and started being automation specialists. The 20-minute budget was enforced as a product requirement — suites that exceed it get parallelized or trimmed — because slow pipelines are where daily deployment quietly dies.

Progressive rollout and the platform team

Deployments became progressive by design. Canary releases expose each change to a slice of traffic first; blue-green switches stage per customer environment; automated rollback hooks wired to error-rate and saturation signals from Datadog revert a bad change in minutes without human intervention. Database migrations follow expand-contract discipline — additive columns first, batched backfills second, cleanup last — so schema changes never block a deploy or require downtime. Six months in, an internal platform team took ownership of the pipeline as a product, with its own roadmap and internal customers, reflecting Gartner's reported prediction that by 2026, 80% of large software engineering organizations will establish platform teams to improve developer experience and throughput. DORA metrics — lead time, change failure rate, MTTR — are now reviewed in the same leadership meetings as feature goals and revenue.

06 The First 90 Days

The first month was measurement and groundwork: the DORA baseline was published, the feature-flag service was integrated, and the first two product teams migrated to trunk-based development as an internal pilot, deliberately chosen because they shipped independently and could validate the workflow without coordinating across the whole organization. Their early experience fed directly back into the rollout — the flag-naming conventions, the trunk hygiene rules and the first version of the automated test templates all came from pilot-team friction rather than from design documents.

Weeks five through eight rebuilt the test pyramid for those pilot teams: contract tests at service boundaries, the integration suite expanded from its skeletal state, and preview environments wired into GitHub Actions so every pull request carried a clickable environment. The first canary deployment ran in week seven against one friendly customer environment, and the automated rollback hook earned its keep within a fortnight — a latency regression that would previously have become a support ticket was reverted in four minutes while the team watched the dashboard. Weeks nine through twelve extended trunk-based development to the remaining product teams in two waves, paired each wave with test-pyramid work, and put expand-contract migration discipline into force for all schema changes. By day 90, deployment frequency had moved from quarterly to weekly, no release weekend had occurred in the quarter, and the 20-minute pipeline budget was holding — the foundation the platform team would inherit and industrialize.

07 The Results

Copperline now deploys daily. Changes reach production in under two days from merge instead of waiting up to a quarter, and the six-week code freeze exists only in retrospect. Change failure rate fell 60% as automated tests and canary rollouts replaced weekend heroics, and MTTR dropped from days to under an hour — a bad change reverts in minutes, triggered automatically, often before customers notice. Release weekends were eliminated outright, recovering thousands of engineering hours a year that had been spent on coordination rather than on customers. The sales team now demos new customer requests shipping within weeks, and that capability showed up in win rates within two quarters.

METRIC	BEFORE	AFTER
Release cadence	Quarterly	Daily
Lead time to production	~90 days	2 days
Change failure rate	Baseline	-60%
MTTR	Days	Under 1 hour

The second-order effects are as consequential as the metrics. Customer trust changed character: instead of quarterly bundles carrying surprises, customers see a steady stream of small, reversible improvements they can

adopt on their own schedule. Engineering morale shifted visibly — release anxiety, once a background hum across 60 people, is gone, and the quarterly merge-war stories stopped being told. Leadership makes capacity decisions from data rather than anecdote, because delivery metrics sit on the same page as revenue and churn. The platform team now treats pipeline reliability the way product teams treat uptime: measured, budgeted and improved continuously.

08 What Your Business Should Do

Copperline's transformation is reproducible if you respect its sequence. Measure your delivery baseline first: record deployment frequency, lead time, change failure rate and MTTR for one full quarter before changing anything, because baseline data makes later progress undeniable and secures funding. Adopt trunk-based development with feature flags — merge small changes continuously and hide unfinished work behind flags — since long-lived release branches are where merge pain and release fear get manufactured. Rebuild your test pyramid deliberately: fast unit suites, contract tests between services and a thin end-to-end layer, because slow, flaky test suites kill daily deployment faster than anything else. Give every branch a preview environment with realistic seeded data so reviewers test actual behavior instead of imagining it from diffs, and QA stops being a queue.

Then make deployment itself safe, because frequency without safety is just faster failure. Deploy progressively with auto-rollback — canary or blue-green every release and wire automatic reverts to real error-rate and latency signals; reverting in minutes is what makes deploying daily rational instead of reckless. Use expand-contract database migrations so schema changes stop dictating your release calendar. Fund a platform team as a product, with its own roadmap and internal customers, rather than treating tooling as a side project; internal users are customers and their feedback deserves the same respect. Finally, review DORA metrics with leadership on the same page as revenue, churn and support load — what leadership reviews monthly is what the organization genuinely works to improve. As you plan, put your own operation to the test: what would you ship if releases carried no coordination cost, how many engineers does your next release mobilize, which customer requests are dying quietly waiting for the next train, how many hours would a critical bug take to reach customers tonight, and can you state your change failure rate from data rather than memory. Work through these eight steps with your team using the action checklist in this practice book, and revisit the reflection questions each quarter — because delivery performance is a habit that needs leadership attention to stay healthy.

REAL-WORLD PROOF & BENCHMARKS

Google's DORA research consistently reports that elite performers deploy on demand, multiple times per day, with change lead times measured in hours rather than months. - DORA Report, 2023

Puppet's State of DevOps research found high-performing IT organizations deploying code 46 times more frequently than low performers, with faster lead times and lower failure rates. - Puppet State of DevOps, 2017

Gartner has reportedly predicted that by 2026, 80% of large software engineering organizations will establish platform teams to improve developer experience and throughput. - Gartner, 2022

THE PRACTICE PAGES

The 8-action checklist

Work through the list with your team. Tick each box as your organization completes the step, and fill in an owner and a target date so every action has a name and a deadline.

☐

1. Measure your delivery baseline first

Record deployment frequency, lead time, change failure rate and MTTR for one full quarter before changing anything. Baseline data makes later progress undeniable and secures funding.

OWNER

TARGET DATE

STATUS

--	--	--

☐

2. Adopt trunk-based development with flags

Merge small changes to trunk continuously and hide unfinished work behind feature flags. Long-lived release branches are where merge pain and release fear get manufactured.

OWNER

TARGET DATE

STATUS

--	--	--

☐

3. Rebuild the test pyramid deliberately

Invest in fast unit suites, contract tests between services and a thin end-to-end layer. Slow, flaky test suites kill daily deployment faster than anything else.

OWNER

TARGET DATE

STATUS

--	--	--

☐

4. Give every branch a preview environment

Spin up ephemeral environments per pull request with realistic seeded data. Reviewers then test actual behavior instead of imagining it from diffs, and QA stops being a bottleneck.

OWNER

TARGET DATE

STATUS

--	--	--

☐

5. Deploy progressively with auto-rollback

Canary or blue-green every release and wire automatic rollback to real error-rate and latency signals. Reverting in minutes is what makes deploying daily rational instead of reckless.

OWNER

TARGET DATE

STATUS

--	--	--

☐ **6. Use expand-contract database migrations**
Split schema changes into additive steps, batched backfills and later cleanups. Non-blocking database migrations end the code-freeze rituals that made every release a coordinated event.

OWNER

TARGET DATE

STATUS

☐ **7. Fund a platform team as a product**
Give a dedicated team real ownership of the pipeline, tooling and developer experience, with its own roadmap. Internal users are customers; treat their feedback accordingly.

OWNER

TARGET DATE

STATUS

☐ **8. Review DORA metrics with leadership**
Put delivery metrics on the same leadership page as revenue, churn and support load. What leadership reviews monthly is what the organization genuinely works to improve.

OWNER

TARGET DATE

STATUS

AFTER THE CHECKLIST: OUR TOP 3 PRIORITIES

From the eight actions above, write down the three you will start first - and the single number you expect each one to move. Three priorities with owners beat eight without any.

PRIORITY 1 NUMBER IT SHOULD MOVE	PRIORITY 2 NUMBER IT SHOULD MOVE	PRIORITY 3 NUMBER IT SHOULD MOVE
--	--	--

THE PRACTICE PAGES

Reflections for your team

Use these questions in your next planning session and type short answers into the boxes. The gaps you find are usually the fastest wins.

Q1. What would your business ship if releases carried no coordination cost at all?

Q2. How many engineers does your next release mobilize, and for how many hours?

Q3. Which customer requests die quietly in your backlog waiting for the next release train?

Q4. If a critical bug ships tonight, how many hours until a fix reaches customers?

Q5. Do your developers trust the test suite — or does everyone quietly test in production?

Q6. What is your change failure rate, and could you state it from data rather than memory?

THE PRACTICE PAGES

My 90-day plan

Finish the session by committing to a plan. Three phases, one goal each, one number that tells you it worked. Bring this page to your next leadership review - or to a free consultation with Emerson Technology, and we will pressure-test it with you.

Days 1-30 Focus: audit and alignment

PRIMARY GOAL**KEY ACTIONS****HOW WE MEASURE SUCCESS****OWNER**

Days 31-60 Focus: build and prove

PRIMARY GOAL**KEY ACTIONS****HOW WE MEASURE SUCCESS****OWNER**

Days 61-90 Focus: scale and measure

PRIMARY GOAL

KEY ACTIONS

HOW WE MEASURE SUCCESS

OWNER

Tip: save a copy per team so each group keeps its own answers. Revisit the reflections every quarter - the questions stay the same even as the numbers move.

BOOK THE REVIEW BEFORE YOU CLOSE THE BOOK

Plans without a review date drift. Before you close this practice book, agree on when you will come back to these pages, who will run that session, and when you will check the numbers again. If you want an outside pair of eyes, bring this page to a free consultation with Emerson Technology and we will pressure-test the plan with you.

REVIEW DATE	REVIEWED BY	NEXT CHECK-IN
------------------------------------	------------------------------------	--------------------------------------

Ready for a transformation like Copperline Software's?

Emerson Technology designs and builds the data, AI, cloud and automation systems behind results like these. On a free consultation call we will map this exact playbook to your business - your data, your constraints, your numbers.

Book a free consultation

support@emersontechnology.com

or start online: emersontechnology.com/contact