ANDREW MCCARRON

Deployment Intelligence

How elite engineering teams predict production incidents before they happen — and why your deploy pipeline is the last place to look

Andrew McCarron · Founder, Koalr

First published March 2026.

Why I Built This

The incident that made a twenty-year consultant start a company

I have spent most of my professional life helping organizations get software out of the door. Twenty years as an IT consultant, working with enterprises ranging from regional banks to global logistics companies to public sector bodies managing critical infrastructure. In that time I have seen thousands of deployments — planned and unplanned, successful and catastrophic, celebrated and quietly buried in incident post-mortems that nobody read twice. I have developed, over those years, a practitioner's intuition for deployment risk. I know, in a way I could not always fully articulate, when a deployment feels dangerous.

The problem is that intuition doesn't scale. It doesn't survive handoffs. It doesn't operate at midnight when the on-call engineer has been woken at 2 AM

and is working from a cognitive reserve that doesn't include twenty years of pattern recognition. It doesn't protect the team that just lost their most experienced senior engineer to a competitor. And it doesn't catch the 47 other deployments happening this sprint that the one experienced engineer with the good intuition never reviewed.

The incident that crystallized this for me happened about fifteen years into my consulting practice. I was advising a mid-sized financial services firm — not a client I can name, but the details are specific enough to still sting. The team was good. The engineers were experienced. The review process was, by industry standards at the time, thorough. And on a Thursday afternoon — not even Friday, which at least would have made it a cliché — a deployment to their customer-facing loan origination service caused a silent data corruption issue that wasn't caught for eleven hours. By the time it was detected, 340

applications had been processed with incorrect risk scores. The remediation took three weeks. The regulatory reporting took longer.

When I did the post-mortem with the team, I found what I always find in these situations: the signals were there. The PR had touched eight files across three modules — high change entropy by any measure. The author was a contractor who had been on the team for three months and had low expertise scores in two of the three modules. The test coverage had declined by 11 percentage points on the changed lines. The service had experienced an incident four weeks earlier that had been only partially resolved. Every one of these facts was sitting in the team's version control system and incident management tool. None of them had been aggregated. None had been surfaced. Nobody had connected the dots.

I spent the next six months trying to find a tool that would connect those dots automatically — that would read the version control history, the incident

database, the review metadata, and surface a coherent risk signal before the deployment, not after. I found pieces: coverage tools, PR size linters, static analysis platforms. I did not find a system. The aggregation layer didn't exist as a commercial product. The research existed — there was a substantial academic literature on just-in-time defect prediction going back to Nagappan and Ball in 2005 and extending through Hassan's change entropy work in 2009 and Kamei et al.'s large-scale validation in 2013. But the research had not made it into production tooling that an engineering team could adopt without a significant internal engineering investment.

That gap — between what the research showed was possible and what teams actually had available — is why I founded Koalr. And this book is the articulation of everything I learned in building the system: the signals that matter, the model architecture that combines them, the organizational

patterns that make adoption succeed, and the cultural prerequisites that make the whole thing stick.

Who This Book Is For

This book is written for three overlapping audiences, and I have tried to serve all three simultaneously.

Engineering managers and VPs of Engineering who are responsible for deployment reliability but currently lack the tooling to manage it proactively. If you have ever sat in a post-mortem and thought "we should have known this was risky," this book gives you the framework and the vocabulary to build a system that catches what your intuition currently catches — and applies it consistently to every deployment your team ships.

Senior engineers and tech leads who understand deployment risk intuitively but want a rigorous framework for systematizing and communicating that intuition. The thirty-three

signals described in Part II are, in large part, a formalization of things experienced engineers already know. The contribution is making them precise, measurable, and combinable into a model that operates consistently at team scale.

Platform and tooling engineers who will actually build the implementation described in Part III and Chapter 14. The Implementation Guide chapter is written specifically for you: it contains working code samples, database schemas, and a week-by-week build sequence that takes you from zero instrumentation to a functioning deployment gate in sixteen weeks.

What This Book Is Not

This is not a book about DevOps practices generally. If you're looking for a comprehensive treatment of continuous delivery, deployment automation, or release engineering, Humble and Farley's *Continuous Delivery* remains the definitive reference and I recommend it without reservation.

This book assumes you are already deploying — probably frequently — and focuses specifically on the pre-deployment risk assessment problem.

This is not a book about incident management or post-mortem culture. Nora Jones and Casey Rosenthal's *Chaos Engineering* and the body of work around SRE practices cover that ground well. This book is about what happens before the incident — the window between "code approved" and "alert fired" where intervention is still possible.

And this is not a book about AI replacing engineering judgment. The system described here uses machine learning as one component of a larger approach that is fundamentally about augmenting human decision-making, not replacing it. The model produces a score. The score informs a human decision. The human retains full authority. The goal is to ensure that when the human makes the deployment decision, they have access to all the

information that is already in their systems — which, as Chapter 3 demonstrates, is more than most teams realize.

A Note on the Research

Every signal in this book has a research foundation. I have cited the primary papers throughout, and the references section contains the full bibliography. The academic literature on software defect prediction is extensive, rigorous, and remarkably consistent: the same signals appear as predictive across different codebases, different languages, different organizations, and different time periods. This consistency is not accidental — it reflects genuine structural properties of software complexity and failure modes that transcend specific technical contexts.

Where I have extrapolated from the research — in the threshold values, the interaction weights, and the organizational implementation patterns — I have been explicit about the distinction between validated

research findings and practitioner-derived recommendations. The signals are well-established. The exact weighting of those signals in any specific organization's model is something that will be tuned by your own data.

A Note on Case Studies

The incident case studies in this book are composites. They are constructed from patterns that appear repeatedly across real engineering organizations and post-mortem databases, but the organizations, teams, engineers, and specific technical details are fictional. No case study should be read as a description of any specific company or incident. The signal profiles, recovery timelines, and financial impacts are representative of real-world patterns but are not drawn from any single event. The preface incident — the financial services loan origination failure — is a partial exception: the

pattern is real, the details have been changed, and the organization cannot be identified from what is written here.

The Acknowledgment I Have to Make

Building Koalr has been a team effort, even in its early stages. The engineers who have helped shape the system, the early design partners who trusted us with their deployment data before we had a polished product, the researchers whose published work forms the intellectual foundation of everything here — this book reflects their contributions as much as mine.

It also reflects the engineers and managers who shared their incident post-mortems with me over twenty years of consulting work. They trusted me with their worst days. I have tried to honor that trust by turning those experiences into a framework that helps other teams have fewer of them.

The financial services firm whose loan origination incident I described above — they went on to build a version of this system internally, after we worked through the post-mortem together. The last time I spoke with their CTO — late 2025 — they had gone fourteen months without a Sev-1 deployment-caused incident. That is what this work is for.

ANDREW MCCARRON

Founder, Koalr · March 2026

00 INTRODUCTION

The Friday Deploy Problem

The call came at 4:47 PM on a Friday.

Sarah, a VP of Engineering at a Series B fintech company, was already gathering her things when her phone lit up with PagerDuty alerts. The payment processing service was down. Not slow — down. Three thousand transactions in flight when the deploy hit, all of them failing with a cryptic 500 error that no one had seen before.

By 5:30 PM, seventeen engineers were on a Zoom call. By 6:00 PM, they had identified the deploy — a seemingly routine update to the payment gateway integration, merged at 4:31 PM by a developer who had been with the company for six months. By 7:15 PM, after a rollback that should have taken fifteen

minutes but took forty-five because the database migration hadn't been reversed, the service was back up.

The next Monday, they ran the post-mortem.

It was a good post-mortem, as post-mortems go. Blameless. Thorough. The timeline was reconstructed with precision: the PR had been created on Thursday afternoon, reviewed by two engineers (neither of whom owned the payment gateway code), approved, and merged on Friday after a third approval from the on-call engineer who had been focused on a separate incident all week and had spent approximately four minutes reviewing the diff.

The root cause was a combination of factors: a database migration that altered the schema of a high-traffic table without a proper lock-free migration strategy, a test suite that covered the happy path but not the migration rollback, and a

coverage blind spot in the payment gateway module that had grown from 73% to 61% over the previous eight weeks without anyone noticing.

Someone asked the question that every good post-mortem produces and most organizations fail to act on:

> *"What would have told us this was going to happen?"*

The answer was uncomfortable. All of it was knowable. The PR had modified a file in the payments module, which was owned by the payments team, and neither reviewer was on the payments team. That was a CODEOWNERS violation — detectable by a machine in milliseconds. The database migration was a DDL change — an ALTER TABLE operation — on a table receiving tens

of thousands of writes per hour. That's a mechanical risk signal, not engineering judgment. The coverage drop from 73% to 61% was visible in Codecov, which the team used but nobody had set up alerts on. The developer who wrote the code had never committed to the payment gateway files before. That's an expertise signal. The merge happened at 4:31 PM on a Friday, which is historically the second-highest-risk deployment window of the week.

- **CODEOWNERS violation** — neither reviewer owned the modified files. Detectable in milliseconds.
- **DDL migration on a high-traffic table** — mechanical risk signal, no judgment required.
- **Coverage drop from 73% → 61%** — visible in Codecov, unwatched.
- **First-time author in this module** — zero prior commits to `src/payments/gateway/`.

- **Friday 4:31 PM merge** — the second-highest-risk deployment window in the week.

Five signals. Every one of them quantifiable. Every one of them available before anyone clicked merge. None of them were surfaced.

This book is about closing that gap.

I call it the *Deployment Intelligence™ gap*: the distance between what was knowable before a deployment and what was actually known. In 2026, as AI coding tools generate pull requests faster than human reviewers can evaluate them, the gap is getting wider, not narrower. The DORA 2025 State of DevOps report found that organizations using AI coding assistance saw a measurable increase in

deployment instability. More code, more PRs, fewer human eyes per line — and the signals we already have, lying dormant in our tooling, going unread.

Deployment Intelligence™ is the systematic, signal-driven approach to closing that gap. It's not a product. It's a discipline — like Business Intelligence, which transformed finance from gut-feel to data-driven decisions, or Revenue Intelligence, which did the same for sales. Engineering operations is next. The signals exist. The data is there. What's missing is the practice of reading it.

What this book is not

This book is not about slowing down. One of the reflexive responses to any discussion of deployment risk is to suggest doing fewer deployments, or smaller ones, or slower ones. That's the wrong prescription. The research on this is unambiguous:

teams that deploy more frequently have fewer incidents, not more. Elite performers deploy multiple times per day and have a change failure rate below 5%. The goal is not to be more cautious. The goal is to be more *intelligent*.

This book is also not primarily about tools. The tooling chapter exists, and I reference Koalr's implementation throughout, but the framework here can be applied with any combination of GitHub, your CI/CD platform, and a few hundred lines of code. The concepts matter more than the implementation.

And this book is not about blaming developers. The deploy that took down Sarah's payment service was not a failure of the engineer who wrote it. It was a failure of the *system* around that engineer — the signals that were present but unread, the governance that existed on paper but wasn't enforced, the expertise gap that was visible in the data but

invisible to the humans making the merge decision. Deployment Intelligence™ is about system design, not individual performance.

What this book is

This book is a practitioner's manual for engineering leaders who want to know, before every deployment, whether it's safe to ship.

Part I establishes why this matters now, more than it ever has. We'll look at the contribution and the limitations of the DORA framework, examine the mechanism by which AI coding tools are widening the deployment instability gap, and do an honest post-mortem on the anatomy of production incidents — what the data shows was knowable before each one happened.

Part II is the technical core: the 33 signals that predict whether a given deployment will cause a production incident, organized into four categories

— structural, historical, contextual, and environmental. For each signal: the mathematical definition, the research backing, how to instrument it, and a real incident where it was the missing predictor.

Part III is the systems chapter: how to combine 23 heterogeneous signals into a single risk score you can act on, how to build the feedback loop that makes the model smarter over time, why CODEOWNERS is the most underused safety system in most repositories, and how to think about the role of LLMs in Deployment Intelligence™ (hint: it's not the role most people assume).

Part IV is about culture. The technical system is the easier half. The harder half is building an organization that uses it — where risk signals inform decisions rather than assign blame, and where the feedback loop actually closes.

Each chapter ends with a one-page summary and questions to answer about your own organization. At the back you'll find the full 33-signal reference card, the Deployment Intelligence™ Maturity Model self-assessment, and a 90-day implementation roadmap.

How to read this book

This book is structured for two types of readers. If you are a practitioner who wants to implement Deployment Intelligence™ right now, Parts II and III are the technical core — everything you need to build the signal infrastructure, train the model, and roll out the Check Run gate. You can read Parts I and IV afterward for context and culture, but the implementation is standalone.

If you are a technical leader making the case for this investment — a VP of Engineering, a CTO, or an engineering manager who needs to convince an organization rather than build a system — Parts I

and IV are where you'll find the most leverage. The business case is in Chapter 15, the maturity framework is in Chapter 12, and the case studies in Chapter 13 provide the specific organizational analogies you'll need for the internal pitch.

If you are neither — if you're a curious reader who wants to understand what state-of-the-art deployment risk management looks like without committing to an implementation project — the introduction, Chapters 1–4, and Chapter 16 provide a complete conceptual overview without requiring you to write a line of code.

The chapters are designed to be read sequentially, but each can stand alone. If you've just had a production incident and want to understand what signals were present, read Chapter 3. If you're about to pitch Deployment Intelligence™ to your CTO, read Chapter 15. If you want to understand why your

team is resistant to risk gates, read Chapter 11. The sequential structure adds context; the individual chapters are self-contained.

A note on the numbers

This book contains specific numbers throughout: AUC values, incident correlation coefficients, false positive rates, ROI estimates. These numbers are real — derived from published academic studies, from the incident databases of organizations that have shared their data for research purposes, or from the composite analysis of case studies presented in Part V. I have tried to be honest about the uncertainty in each number and to document its source.

That said: the numbers in your organization will differ. The 400-line PR size threshold at which defect detection drops below 20% is an empirical average across hundreds of code reviews at

technology companies. Your threshold may be 300 lines or 500 lines, depending on your codebase's complexity, your team's review culture, and the nature of your engineering work. The 0.74 standalone AUC for change entropy is an average across multiple open-source projects; your codebase may show higher or lower predictive power for that specific signal depending on how changes are structured in your repository.

The right response to this uncertainty is not to distrust the numbers — they are the best available evidence for the underlying relationships. It is to treat them as starting points, calibrate them against your own incident data, and update them as your model accumulates training examples from your specific engineering context. The framework is universal. The calibration is always local.

The discipline in formation

Deployment Intelligence™ is an emerging discipline. The signals in this book are validated. The model architecture is proven. The organizational patterns — phased rollout, Risk Review, developer experience design — are derived from real implementations that have succeeded. But the field is young, and the frontier is moving.

This book represents the state of the discipline as of early 2026. Three years from now, there will be chapters that need to be rewritten: IDE behavioral signals will be standard taxonomy. Semantic change analysis using LLMs will have matured from experimental to production-grade. Federated cross-organization models will exist. The specific implementation tools will have evolved. The core framework — the four signal families, the phased rollout, the accuracy retrospective, the non-

negotiable boundary about non-use in performance review — will not change. The signals improve. The principles endure.

What I hope this book does is give the practitioners who are building these systems today the vocabulary, the framework, and the evidence they need to build them well — and to make the case to the organizations around them that this work is worth doing. The deployment that didn't happen is the highest-value outcome of this discipline. It is also invisible. It is celebrated by no one, because no incident happened and no one knows what was prevented. The engineers who do this work do it without the recognition that comes from visibly heroic incident response. They are doing something harder and more important: preventing the incident that would have required the heroism.

That work deserves a framework. Here is one.

Let's start with a question.

The question is always the same: what was knowable before this deployment that would have changed what we did? Everything in this book is an attempt to answer it systematically, at scale, before the merge button becomes available rather than after the alert fires.

PART I

Why We Deploy Blind

And why that's no longer acceptable

01 CHAPTER ONE

DORA Told Us What Happened. Nobody Told Us What Was About to Happen.

In 2018, Nicole Forsgren, Jez Humble, and Gene Kim published a book that changed how engineering leaders think about their work.

CONTINUE READING

The book continues with 16 chapters and 32 validated deployment signals.

The Community Edition (free, Parts I & II) and full edition are both available on Leanpub.

leanpub.com/deploymentintelligence