EXECUTIVE BRIEFING

KOHL DIGITAL

Engineering AI for Production

Lessons From Mission-Critical Systems

How to ship AI the business can depend on: five controls, a split between code and model, failure planning, change management and what to monitor after launch.

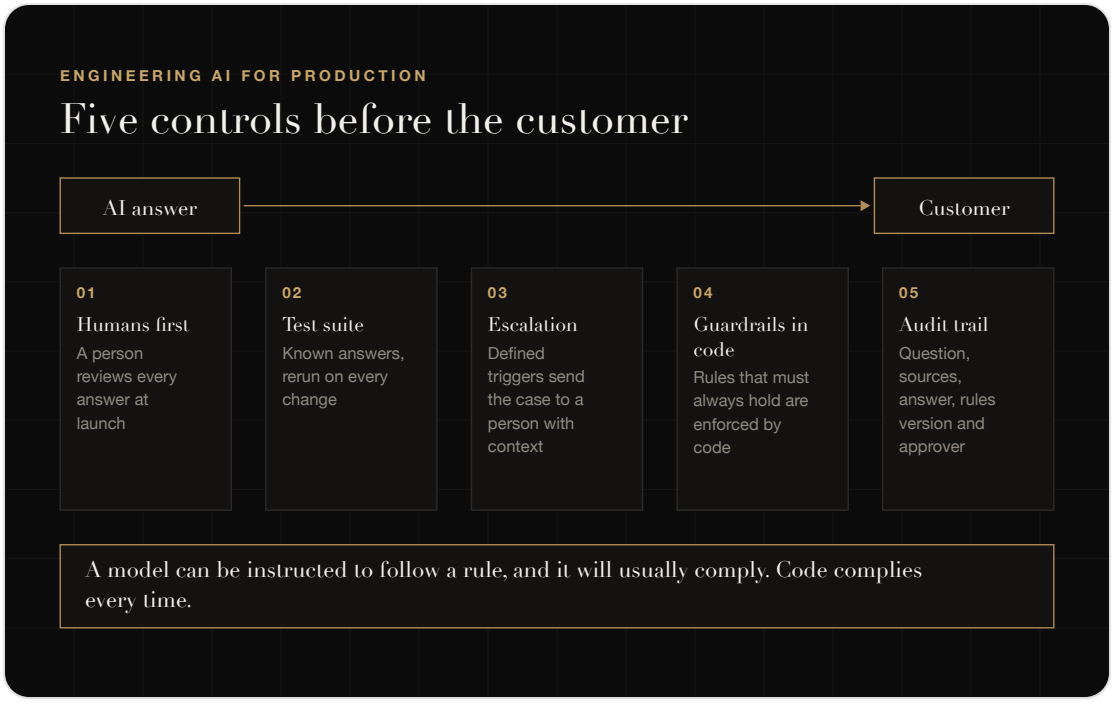
Mike Kohl
Founder, Kohl Digital

mikekohl.com/guides/engineering-ai-for-production

KEY TAKEAWAYS

- AI fails quietly with confident wrong answers, so ship it with five controls: humans first, tests, escalation, guardrails in code, audit trail.
- Give code the jobs with one correct answer, such as prices, policy limits and access checks, and give the model classifying and drafting.
- Version prompts, rules and knowledge records, rerun the full test suite after every change, and monitor weekly accuracy samples, escalation rate and source freshness.

You ship AI the business can depend on by treating it like any other production system, with one addition: you assume it will sometimes be fluent and wrong, and you build controls that catch that before a customer does. Five controls do most of the work: humans first, a test suite on known answers, escalation by design, guardrails in code and an audit trail.



Five controls stand between every AI answer and the customer.

This guide is written for the CTO or head of engineering who owns the number that matters after launch: the share of AI output that is correct, and the cost of the share that is not. It draws on more than 20 years of enterprise software delivery, including systems where failure had consequences.

Why AI fails differently

Traditional software fails loudly. A service goes down, an exception fires, a page turns red. Your monitoring was built for that.

AI fails quietly. A model returns a confident, well-written answer that is wrong, and nothing in your stack flags it. No error code, no spike in latency. The failure shows up weeks later as a customer complaint, a wrong quote or a policy that was misstated.

That changes what "tested" means. A demo that answers ten questions well proves very little. You need evidence that the system is right on the questions your business actually receives, and a way to see when it drifts.

It also changes where you spend engineering effort. In a conventional project, most of the effort goes into making the system do the thing. In an AI project, a large share goes into proving it does the thing, and into deciding what happens when it does not.

What I learned building systems that were not allowed to fail

Early in my career I built message-switching software at Computer Projects of Illinois that served the FBI, the DOJ and state law enforcement agencies. Messages had to arrive, in order, every time. The lesson I took from that work, and still apply, is to assume every component will fail and decide in advance what happens when it does.

I also did engineering work for the Department of the Interior under federal security clearance. That kind of work builds habits: written procedures, limited access, records of who did what.

Those habits transfer directly. AI is a new component with an unusual failure pattern, and the answer is the same discipline: define the failure, plan the response, test the plan.

The five controls

These controls are the same ones that sit behind the AI knowledge system my team built for a client. It was built from 7,519 historical customer question-and-answer records. Before launch it had to pass a 370-question QA suite of real questions with known answers, and it passed with zero errors. It now answers more than half of the customer questions that used to reach staff.

Here is what each control means in practice.

1. Humans first

At launch, a person reviews every answer before it reaches a customer. This is the starting position, and it produces the data you need: which answers were accepted, which were edited, which were rejected, and why.

Relax review in steps, by question type, only after the record supports it. Reviewers come from the people who already know the answers, so their corrections go back into the system as new rules.

2. A test suite on known answers

Collect real questions with answers your experts have confirmed. Run the system against them. Define the pass bar before you run the suite, and write it down.

The suite is an asset you keep. It grows as you find new failures, and every future change runs against it.

3. Escalation by design

The system needs a defined path for everything it should not answer. Decide the triggers in advance: low confidence, a topic outside its scope, an angry customer, a request that touches money or legal terms.

A good escalation sends the case to a person with the context attached, so the person does not start from zero. Multiple layers help: a first check, a second check, then a human.

4. Guardrails in code

Anything that must always be true belongs in code, where it cannot be argued with. A model can be instructed to follow a rule, and it will usually comply. Code complies every time.

Examples: a discount ceiling, a list of topics the system may never discuss, a requirement that every answer cite a source record, a block on any write to a system of record without approval.

5. An audit trail

Record the question, the sources retrieved, the answer given, the version of the rules and prompts in force, and who approved or edited the result. When someone asks "why did it say that?", you can answer in minutes.

The audit trail also feeds improvement. Patterns in edits and escalations show you where the knowledge is thin.

Deterministic parts and model-driven parts

The most common architecture mistake is giving the model jobs that code should do. Split the system deliberately.

Job	Belongs to	Why
Calculating a price, discount or date	Code	The answer has one correct value
Enforcing a policy limit	Code	The rule must hold every time
Checking who may see what	Code	Access is a yes or no decision
Looking up a record by ID	Code	Exact match, no interpretation
Classifying what a customer is asking	Model	Language varies widely
Drafting an answer from approved sources	Model	Wording flexes, facts come from the source
Summarizing a long thread for a person	Model	A human checks the summary
Deciding whether to escalate	Both	Model proposes, code applies the rules

On the enterprise sales platform I architected at Level 3 Communications and Lumen Technologies, the rules for each product line were held in data instead of code, across renewal systems spanning more than 1,000 product lines. The same idea applies here. Put facts and rules where they can be read, versioned and audited. Let the model do the language work around them.

Plan for failure before launch

Take every component and ask the same three questions.

1. How will we know it failed?

2. What does the customer or employee see when it does?
3. Who is notified, and what do they do in the first ten minutes?

Components include the model provider, the retrieval layer, your knowledge sources, the integrations into other systems and the people in the review queue.

Run the failure scenarios as drills before launch. Turn off the model connection and watch what happens. Feed in a corrupted source record. Send a question in a language you did not plan for. Each drill shows whether the fallback works or only exists on a diagram.

A failure plan table

Fill in one row per component. A short version looks like this.

Component	Failure signal	Fallback	Owner
Model provider	Timeouts or error responses	Route to human queue, show a plain status message	Platform lead
Knowledge source	Record older than its review date	Flag the answer, route to the record owner	Knowledge owner
Retrieval layer	No source found for a question	Escalate, never improvise	Engineering
Review queue	Items waiting beyond the agreed time	Alert the team lead, pause new auto-answers	Operations lead

Change management for a system that keeps changing

An AI system has more moving parts than most software: the model version, the prompts, the rules, the knowledge records and the retrieval settings. Any of them can change behavior.

Treat all of them as versioned artifacts.

- Keep prompts and rules in version control, with a named owner for each.
- Run the full test suite after every change, including changes that look trivial.
- Record which version produced each answer, so the audit trail links back to the exact configuration.
- Roll out changes to a small share of traffic first, and keep the previous version ready to restore.
- When your model provider updates a model, run the suite again before the new version goes live.

A test suite that runs only at launch gives you a snapshot. Run it on every change and you have a standing guarantee.

What to monitor in production

The signals that matter are quality signals, and they will not appear on a standard infrastructure dashboard. Set up at least these.

- **Accuracy on a sample.** Pull a set of live answers each week and have an expert grade them against the sources.

- **Escalation rate.** Track the share of questions sent to people, and the reasons. A sudden drop can be as worrying as a spike.
- **Edit and rejection rate.** While humans still review, this tells you how close the system is to being trusted for each question type.
- **Source freshness.** How many answers rely on records past their review date.
- **Repeat questions.** The same question asked again after an answer often means the answer did not help.
- **Cost and volume per workflow.** Unexpected growth is a clue that something upstream changed.

Set a threshold for each signal and an owner who acts when it is crossed. A metric with no threshold and no owner is decoration.

A first-month procedure

If you own an AI system heading to production, run this sequence.

1. Write down the workflow, the number it moves and the person who owns that number.
2. Collect at least a few hundred real questions or cases, with expert-confirmed answers. Set the pass bar.
3. List the facts and rules the system must respect. Mark which go in code and which go in the knowledge base.
4. Define the escalation triggers and the person who receives each one.
5. Build the audit trail before you build the interface.
6. Run the failure drills from the table above.

7. Launch with humans reviewing every answer. Review the record weekly.
8. Relax review one question type at a time, only where the record supports it.

Three guides go deeper on the questions engineering leaders raise next.

- Read the guide on AI governance for mid-market companies
- Read the guide on moving from AI pilot to production
- Read the guide on adding AI to legacy systems

The facts your system answers from deserve the same care as its code. A company brain gives every fact a source, an owner and a date, keeps history separate from current truth, and applies your access rules.

What goes wrong

Most production failures come from a short list.

- **The demo became the test.** The team shipped after checking a handful of questions they chose themselves. The first week of real traffic found the gaps.
- **Rules lived in the prompt.** Someone updated a pricing rule in a prompt and nobody reran the suite. The system started quoting an old discount.
- **No owner for a source.** A policy changed, the knowledge record did not, and the system kept repeating the old policy with full confidence.
- **Escalation existed on paper.** The path to a human went to a shared inbox no one watched.
- **The model wrote to the system of record.** An unreviewed change landed in production data and took days to find.

- **Review stopped too soon.** The team removed human review across all question types after a good first month, and a rare category went unchecked.
- **No version record.** A customer disputed an answer and nobody could say which prompt or rules produced it.

Questions leaders ask

How accurate does the system need to be before launch?

Set the bar by the cost of a wrong answer in each question type. A wrong store-hours answer and a wrong contract term are different problems. Write the bar down before testing, and launch with human review on every answer so a miss has a safety net.

Can we use a general model without building any of this?

You can use one for personal drafting and low-stakes work. For anything that speaks to customers or changes records, the controls apply, because the failure pattern is the same whichever model you pick. The model is one component and the controls are what make the system dependable.

Who should own the system after launch?

One named person owns the outcome number, and the knowledge sources each have an owner. Engineering owns the platform. The business owns the facts. When ownership is split without names, the first failure takes days to route.

When can we remove human review?

When the record shows consistent correctness for a question type, over enough volume that you trust the sample, and when escalation and guardrails are tested. Remove review one type at a time and keep sampling afterward.

The Leverage Diagnostic takes about six minutes, scores the six places leverage hides and prices the gap with the math shown. Start with the Leverage Diagnostic to see where your production risk sits.

NEXT STEP

Find where leverage hides in your company.

The Leverage Diagnostic takes about six minutes, scores the six places leverage hides and prices the gap with the math shown.

mikekohl.com/diagnostic

Kohl Digital builds AI systems for mid-market and enterprise companies. Founder Mike Kohl architected the enterprise sales platform at Level 3 Communications and Lumen Technologies that went from $0 to $500 million in four years.

Book a call: mikekohl.com/contact