IcebergQA

A FIELD GUIDE

You Didn't Write That Code. You Still Own It.

What AI-assisted development actually means for the people responsible when something breaks.

By Phil Lew · IcebergQA

icebergqa.com · testers.ai × XBOSoft

THE PROMPT IS GONE

I was talking with an engineering manager at a 25-person company a few months ago. Smart team. Moving fast. They had adopted AI coding tools early and were genuinely proud of how much they were shipping.

I asked him one question: when something breaks in production, how long does it take to understand why?

He paused longer than I expected.

The problem wasn't that his team was doing anything wrong. The problem was that a meaningful portion of their codebase had been generated by AI tools, reviewed quickly, tested against the old pass/fail suite, and shipped. Nobody on the team fully understood the logic that assembled it. When behavior in production turned out to be unexpected, nobody opened the code to find out why.

They fed the problem back into the model and generated something new.

> *That is not a fix. That is stacking a second unknown on top of the first one.*

Sometimes the new version works. Sometimes it breaks something else two weeks later and nobody can say which prompt caused it.

If that doesn't sound urgent yet, it should. This guide exists because most companies treat this as a process question when it is actually a hair-on-fire problem. The gap between how fast AI lets you ship and how much your team actually understands about what shipped is not closing. It is widening every sprint.

And the people who eventually own the consequences are not the model. They are you. How many times can you say: *the AI did it.*

That may work once if at all. If you use that excuse, more than likely, you'll be fired.

Your Software Changed. Your Testing Didn't.

There is a straightforward assumption baked into most testing processes: give the software an input, check the output, decide if it matches what you expected. That model is not wrong. A login form should still reject a bad password. A checkout flow should still charge the right amount.

But when you use AI to generate your code, that is no longer the whole picture.

When developers use AI to write code, they introduce behavior that is harder to pin down than a traditional function. AI-assisted code can behave differently across runs. It can produce outputs that

are technically correct but factually wrong. It can pass a test that checked the wrong thing.

> *A non-deterministic system is one where the same input may produce different outputs. Sometimes that variation is intentional. Sometimes it is a side effect of timing, randomness, or hidden state. Sometimes it is a bug.*

The hard part is that those three things can look identical from the outside. Your test suite may not tell you which one you are dealing with.

Most QA processes at small and mid-size companies were not designed for this. They were designed for a world where the developer wrote the code, understood it, and handed it to QA to verify against known behavior. That handoff looks very different now. Developers are using AI to write production code, explain the code, review the code, and generate tests around the code. They are not waiting for QA to catch up.

> *The gap between what shipped and what was verified is quietly widening at a lot of companies right now.*

Quiet is the dangerous part. Nobody sounds an alarm when a gap widens slowly. It just shows up later as an incident report.

Not All Differences Are Bugs. Some Are Lies.

Here is a distinction that matters more than most people realize.

When a non-deterministic system produces different outputs, not all of that variation is a problem. Some variation is harmless. If your support tool says “Your refund was approved” one time and “We approved your refund” another time, the user gets the same information. The wording changed. The truth did not.

But some variation changes the truth.

If your system tells one user that returns are accepted within 30 days and tells another user 45 days, that is not a style difference. That is two different policies. One of them is wrong. And your test suite, if it was checking output strings rather than factual accuracy, probably passed both.

> *The tester’s job is not to eliminate all variation. The tester’s job is to understand which variation is acceptable and which variation is dangerous.*

This is the distinction your old testing process was not built to make. Exact string comparisons are powerful when correctness is exact. But AI-generated outputs do not conform to expected strings. They conform, sometimes, to expected truths. And those are different things to verify.

For a founder or CTO, this surfaces as a specific kind of risk. Your billing logic, your access controls, your refund handling, your user-facing policies, any of it built with AI assistance and tested only against exact output expectations, may already be telling different users different versions of the truth in production right now.

> *The green light is real. It is just checking the wrong question.*

Pass/Fail Is Too Blunt for This World.

Pass/fail testing is not wrong. It is just insufficient on its own.

An AI-generated answer may be correct but vague. A recommendation may include useful results but miss the most important one. A generated summary may be accurate but too long to be useful. Calling all of these simply pass or fail loses the information you actually need.

What replaces it? Evaluation criteria. Properties that define acceptable behavior rather than expected strings.

Instead of asking whether the output matches a sentence, you ask whether the output satisfies the requirements that matter: Does it communicate the correct policy? Does it avoid inventing exceptions? Is it clear and direct? Does it point the user toward an appropriate next step?

Those criteria can be checked by humans, by automated rules, or by a combination. The important thing is that the test now matches the real quality question instead of a brittle string comparison that will fail the right answer and pass the wrong one.

> *A system with scores of 8, 8, 8, 8 is very different from one with scores of 10, 10, 4, 8, even if the averages look the same.*

That last point is something we need to think about. Average quality is not the same as consistent quality. A system that is mostly excellent but occasionally catastrophic is not an 8 out of 10. It is a liability wearing an 8's clothing. The tail behavior, the worst-case outputs, the 4 hiding inside an otherwise healthy average, that is where your real risk lives, and it is invisible until it isn't.

This is what a meaningful quality gate looks like for AI-assisted systems: an average quality threshold, a floor below which no output should fall, and a hard stop on any output that crosses into

unsafe or seriously misleading territory. Not just “did it pass” but “how bad can it get.”

The Judge Problem.

If you accept that testing AI-assisted code requires evaluating properties rather than checking strings, you immediately run into a scaling problem. You cannot have a human read every output from every feature before every release. Too much.

This is where AI-based evaluation comes in. An LLM judge is a model used to evaluate another system’s output. You give it the input, the output, the relevant policy or source material, and a rubric. It scores the output and explains its reasoning.

This can scale evaluation in ways that manual review cannot. A judge can compare outputs, cluster common failure patterns, flag borderline cases for human attention, and explain why a particular answer looks risky. It makes it possible to evaluate at a volume that would otherwise be impossible.

But there are three things a judge cannot do for itself.

- It cannot define what quality means for your specific product. A support assistant and a medical summarization tool need completely different rubrics. Generic criteria produce generic evaluations.
- It cannot calibrate itself. Without human oversight of the judge’s decisions, you do not know whether the judge is grading too harshly, too generously, or missing the failure mode that matters most to you.
- It cannot replace the human who knows what the product is actually supposed to do. That knowledge has to come from somewhere.

The human is not optional. The human is the part that makes the whole system trustworthy.

When someone who actually understands software quality sits down and works through an evaluation properly, they beat most AI tools available today. Not because the AI is weak. Because judgment about what matters for a specific product, in a specific context, for a specific user, doesn’t come pre-installed in any model.

Pull the human out to save time and you haven’t removed the risk. You’ve removed the only thing that was catching it.

You Can’t Say the AI Did It.

There is a version of this conversation that is about testing methodology. Forget that version.

This is about accountability, and accountability does not care how the code got written.

When a user is screaming, when a regulator is asking questions, when your board wants to understand what happened, “the AI wrote it” is not an answer. It was never going to be an answer. The developer reviewed it. The team shipped it. The CEO signed off on the release. Ownership does not transfer to the model, no matter how badly anyone in the room wishes it would.

This is the part that a lot of fast-moving companies have not fully reckoned with yet. AI has lowered the cost of writing code.

> *AI does not lower the cost of a production failure. AI does not transfer legal exposure. AI does not reassign the relationship with your user.*

What it does is change how you have to think about being ready to ship.

Being ready used to mean: we tested it, the tests passed, QA signed off. That process was built for a world where the developer wrote the code and could, in principle, explain every line of it.

Being ready now means something different. It means understanding which parts of your codebase behave non-deterministically and why. It means having evaluation criteria that match what your product actually promises. It means knowing your tail risk, not just your average quality. It means having a human who can look at the system and say, with genuine confidence, that this is ready.

> *AI is shipping code faster than most teams can trust it. That gap is where the risk lives.*

The companies that figure this out first are not going to slow down. They are going to ship just as fast with something the others do not have: actual confidence in what they built.

What Good Looks Like

You do not need a massive QA operation to do this well. Most of the companies we work with are small and moving fast. That is not the constraint.

The constraint is usually that nobody has sat down and asked the right questions before the release. What can vary in this system? What variation is acceptable and what is dangerous? What does a failure look like for this specific product? What would a 4 out of 10 mean here?

Once you can answer those questions, a lot of the rest follows. You can build evaluation criteria that match your product. You can set release gates that give you genuine confidence rather than a green dashboard you cannot interpret. You can have a conversation with your team about risk that is

grounded in something real.

The next generation of software quality is not about slowing down. It is about measuring the right things, understanding uncertainty rather than hiding it, and having humans in the loop who can tell the difference between harmless variation and a lie hiding in the output.

The teams that figure that out will ship faster than the ones still running string comparisons against AI-generated code and calling it tested.

Everyone else will find out the hard way, in production, in front of a user, with no good answer for how it happened.

Before your next release, not after.

IcebergQA pairs AI-powered testing infrastructure with experienced human QA judgment, built specifically for teams shipping AI-assisted code faster than their old QA process can verify it. If any part of this guide made you think about your own next release, that’s worth a conversation before you ship it, not after.

icebergqa.com · The 30-Day AI QA Rescue Sprint