

Founder's Pre-Diligence Self-Audit

Diligence isn't an exam you can cram for. It's a survey of decisions you made over years, read by someone paid to be sceptical.

The good news is that almost nothing on this list is a surprise to the team. Engineers know where the bodies are. What kills deals isn't the debt, it's the debt nobody disclosed, found by a stranger, in week three, after you said the architecture was solid.

Run this at least a quarter before you expect to raise or sell. Anything you can fix, fix. Anything you can't, write down and disclose early. Disclosed problems get priced. Discovered problems get punished.

How to use this honestly. Have an engineer answer, not the founder. Then have a second engineer disagree with the first. The gap between those two answers is roughly what diligence will find.

The five things they check first

OK CON N/A

- ☐ ☐ ☐ Can you explain your architecture in plain English, in under five minutes, without a diagram you made for this meeting?
- ☐ ☐ ☐ Is there code in production older than six months, or have you rewritten everything twice?
- ☐ ☐ ☐ Do you deploy at least weekly?
- ☐ ☐ ☐ Can you describe your last outage and what changed because of it?
- ☐ ☐ ☐ Do you know how long a rollback takes, because you have done one?

What you will be asked to produce

Have these ready. Producing them slowly is itself a finding, and it's the cheapest finding to avoid.

OK CON N/A

- ☐ ☐ ☐ Repository access with full history
- ☐ ☐ ☐ An architecture diagram an engineer drew
- ☐ ☐ ☐ A dependency list or generated SBOM
- ☐ ☐ ☐ Twelve months of cloud bills
- ☐ ☐ ☐ Incident log or postmortems
- ☐ ☐ ☐ Any penetration test or audit, including unflattering ones
- ☐ ☐ ☐ Signed IP assignment agreements for everyone who wrote code
- ☐ ☐ ☐ Open source policy and a recent scan
- ☐ ☐ ☐ Model cards and data provenance, if you ship ML

Ownership: the things that void a deal

Fix these first. They're binary, they're unglamorous, and they're the ones that stop a transaction rather than reprice it.

OK CON N/A

- ☐ ☐ ☐ Every employee who wrote code has a signed IP assignment
- ☐ ☐ ☐ Every contractor does too, including the agency you used in year one
- ☐ ☐ ☐ Code written before the company existed has been assigned to the company
- ☐ ☐ ☐ Domains, trademarks and app store accounts are held by the company, not a founder
- ☐ ☐ ☐ No copyleft code is linked into your proprietary codebase
 - GPL or AGPL in the wrong place can force disclosure of your source. That removes most of what you're selling.
- ☐ ☐ ☐ You know which licences are in your dependency tree, from a scan and not from memory
- ☐ ☐ ☐ There's a written policy for adding a dependency
- ☐ ☐ ☐ Anything your team contributed to outside projects went through an approval step

Concentration risk

OK CON N/A

- ☐ ☐ ☐ No single person is the only one who understands a critical system
- ☐ ☐ ☐ Knowledge is written down somewhere a new hire can find
- ☐ ☐ ☐ A new engineer is productive in two to four weeks, and you have recent evidence
- ☐ ☐ ☐ Nobody critical is about to vest out and leave
- ☐ ☐ ☐ The founder isn't the only person who can deploy

The codebase as a stranger sees it

OK CON N/A

- ☐ ☐ ☐ Git history looks like steady work, not three heroic weekends
- ☐ ☐ ☐ No single file changes on every commit
- ☐ ☐ ☐ Tests exist on the paths that would embarrass you if they broke
- ☐ ☐ ☐ Tests actually fail when the code breaks, and you have checked recently
- ☐ ☐ ☐ CI runs on every commit and the team doesn't routinely override it
- ☐ ☐ ☐ The README works on a clean machine
- ☐ ☐ ☐ Architecture decisions are recorded somewhere, even briefly
- ☐ ☐ ☐ You can name your top three areas of technical debt without looking

AI-assisted code

OK CON N/A

- ☐ ☐ ☐ You know roughly how much of the codebase was AI-generated
- ☐ ☐ ☐ Someone on the team can explain the AI-heavy parts without the assistant open
- ☐ ☐ ☐ Generated code was reviewed by a human, and the review is visible in history
- ☐ ☐ ☐ Tests around generated code were written by a person
- ☐ ☐ ☐ You have debugged a production issue inside generated code at least once
- ☐ ☐ ☐ If a model runs in production, you know the cost per request and what happens on an outage

Operations

OK CON N/A

- ☐ ☐ ☐ You can answer “what if the primary database dies” with a procedure, not a hope
- ☐ ☐ ☐ Backups are tested by restoring them, on a schedule
- ☐ ☐ ☐ Rollback is tested and timed
- ☐ ☐ ☐ Secrets are not in the repository, including in the history
- ☐ ☐ ☐ Credentials can be rotated without downtime
- ☐ ☐ ☐ Access is scoped, and not everyone is an admin
- ☐ ☐ ☐ There's an incident runbook, even a short one

Money

OK CON N/A

- ☐ ☐ ☐ Cloud spend is proportional to revenue and you can show the trend
- ☐ ☐ ☐ You know cost per user or per transaction
- ☐ ☐ ☐ No forgotten environments running at full size
- ☐ ☐ ☐ Committed spend agreements are documented with end dates
- ☐ ☐ ☐ You can explain any line of the bill that grew faster than usage

Disclose, don't hide

OK CON N/A

- ☐ ☐ ☐ List everything on this page you failed
- ☐ ☐ ☐ For each, write one line: what it is, what it costs to fix, when you will
- ☐ ☐ ☐ Decide now what you will volunteer in week one

—Volunteering a known problem converts it from a discovery into a plan. It's the cheapest credibility you can buy.

- ☐ ☐ ☐ Make sure your CTO and your deck agree with each other
- ☐ ☐ ☐ Make sure your team gives the same answer you do

The honest question

OK CON N/A

- ☐ ☐ ☐ If a competent stranger spent two weeks in your systems, what would they find that you haven't told anyone?
- ☐ ☐ ☐ What would it cost to fix that before they look?
- ☐ ☐ ☐ Is that less than what it will cost you in valuation if they find it?