

FOR FOUNDERS WITH A LIVE APP

Is your engineering *under control*?

A plain-language health check to run with your team, agency, or freelancer. Seven areas, the questions to ask, the evidence to request, and the warning signs that mean something needs an owner.

	SOLID	SHAKY	UNKNOWN
01 Ownership	☐	☐	☐
02 Production visibility	☐	☐	☐
03 Critical user journeys	☐	☐	☐
04 Release reliability	☐	☐	☐
05 Maintenance	☐	☐	☐
06 Recovery	☐	☐	☐
07 Access to essential accounts	☐	☐	☐

Ask for evidence, *not reassurance*.

WHO SHOULD BE IN THE ROOM

You, and whoever runs engineering: a CTO, lead developer, agency, or freelancer. Set aside about an hour.

WHAT YOU LEAVE WITH

A read on all seven areas, a list of evidence you still need, and no more than two next actions, each with an owner and a date.

INSIDE

How to use it 2
The seven areas 2 to 5
Account inventory 6
Next actions 6

Engineers are *the dam*.

A healthy engineering team holds back a steady flow of small failures you never see: library updates, OS releases, store policy changes, expiring certificates, slow database queries, migrations. When the dam holds, nothing happens, so nobody notices the work. Founders usually find out it cracked the worst way, from a one-star review.

You don't need to read code to know whether engineering is under control. You need to ask direct questions, ask to see the evidence, and notice when the answer is a shrug. The goal is not to turn you into an engineering manager. It is to make sure engineering has a clear owner, so your time goes back to customers, partnerships, and growth.

Ken Vermeille, Founder, Vermillion

FOUR STEPS

- 01 Book an hour with whoever runs engineering. Tell them the goal is to find gaps, not to grade people.
- 02 Go area by area. Ask the questions. Ask to see the evidence, not a description of it.
- 03 Mark each area Solid, Shaky, or Unknown.
- 04 On page 6, choose no more than two next actions, each with one owner and a date.

HOW TO READ EACH AREA

Solid

You saw the evidence and it holds up.

Shaky

It exists, but it is partial, out of date, or depends on one person.

Unknown

Nobody could show you. That is a finding, not a failure.

There is no score and no passing grade. This check shows you where to look. It does not rank you against anyone.

AREA 01 Ownership

Design, engineering, analytics, and monetization overlap more every year, and everyone on the team owns part of the product. But one person still has to be the head chef: the one who makes the final call on the technical system and answers for the outcome.

QUESTIONS TO ASK

- Who is accountable for the app working in production? One name.
- Who decides what gets built, and who decides how?
- When something breaks at night or on a weekend, who hears about it, and who fixes it?
- If our lead developer left tomorrow, what would we lose?

EVIDENCE TO REQUEST

- ☐ A one-page list of who owns what: apps, backend, releases, vendors, accounts
- ☐ The escalation plan for outages, even an informal one
- ☐ Setup notes a new developer could follow to run the app

WARNING SIGNS

- The answer to "who owns this" is a group, a vendor, or you.
- Critical knowledge lives in one person's head.
- You coordinate developers day to day.

Your read

☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

AREA 02 Production visibility

If the team can't see what real users are experiencing, it is guessing. You want to hear about problems from your tools before you hear about them from reviews.

QUESTIONS TO ASK

- If the app started crashing for some users this afternoon, how would we know, and how fast?
- What is our crash-free rate today? What are our slowest important screens and requests?
- Who looks at these numbers, and how often?
- What alerts exist, and who receives them?

EVIDENCE TO REQUEST

- ☐ The crash reporting dashboard (Sentry, Crashlytics, or similar), last 30 days
- ☐ Real usage data from analytics, including session recordings if you have them (PostHog or similar)
- ☐ Backend errors and slow requests. P99 latency, for example, shows how slow things are for the unluckiest 1 in 100 requests.
- ☐ One recent alert, and what happened next

WARNING SIGNS

- The team learns about bugs from you, from support, or from reviews.
- Nobody can pull up the crash dashboard in under a minute.
- Alerts land in a channel or inbox nobody reads.

Your read

☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

AREA 03 Critical user journeys

A handful of paths pay the bills: signing up, reaching the moment of value, paying, restoring a purchase, logging back in. They are your core loop in practice. They should be written down, checked before every release, and watched in production.

QUESTIONS TO ASK

- What are our five most important user journeys? Are they written down?
- Which are tested before every release? Automatically or by hand?
- When did someone last buy and restore a subscription on a real device, on both platforms?
- What happens to a user who hits an error on the paywall?

EVIDENCE TO REQUEST

- ☐ The written list of critical journeys
- ☐ Test results or the checklist from the last release
- ☐ Purchase and restore activity from your subscription platform (RevenueCat or similar)
- ☐ A funnel for each journey in your analytics tool

WARNING SIGNS

- Nobody can list the critical journeys without stopping to think.
- Payment gets tested by paying customers, in production.
- A recent release broke sign-up, login, or purchase.

Your read

☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

AREA 04 Release reliability

Shipping should be boring. When every release feels risky, teams ship less often and bundle more changes together, which makes each release riskier.

QUESTIONS TO ASK

- Walk me through how a release ships. Could someone besides our lead do it?
- If we found a serious bug today, how soon could a fix reach users?
- When a release goes wrong, can we pause it? How do we undo it?
- How often do we ship, and how often does a release cause a problem?

EVIDENCE TO REQUEST

- ☐ The release process, written down
- ☐ Release history for the last three months, including hotfixes
- ☐ Use of phased release on the App Store and staged rollouts on Google Play
- ☐ An automated build pipeline (CI) that builds and tests every change

WARNING SIGNS

- Releases depend on one person's laptop.
- Nobody can tell you what is in the next release.
- Hotfixes are routine, or store rejections catch the team by surprise.

Your read

☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

AREA 05 Maintenance

This is the invisible work. Google Play raises its target API level requirement every year, and Apple periodically raises the minimum Xcode and SDK versions for App Store submissions. Libraries go stale, SDKs get retired, and privacy and payment rules change. None of it shows up in a roadmap meeting, and all of it breaks the app if it is ignored.

QUESTIONS TO ASK

- Which Apple and Google deadlines affect us in the next six months?
- How far behind are we on our framework and key libraries? For React Native apps, which version are we on?
- How much engineering time goes to maintenance? Is it planned or squeezed in?
- What have we been putting off, and what happens if we keep putting it off?

EVIDENCE TO REQUEST

- ☐ A dated list of upcoming platform and policy requirements
- ☐ A report of outdated or unsupported dependencies
- ☐ Maintenance tickets in the tracker, visible next to feature work
- ☐ Recent invisible wins: upgrades, speedups, and tools that made the team faster

WARNING SIGNS

- Maintenance only happens when something breaks.
- "We can't upgrade that" is a common answer.
- A store deadline caused a scramble in the last year.

Your read

☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

AREA 06 Recovery

Things will break. What matters is how fast you notice, how fast you recover, and whether the same thing breaks twice.

QUESTIONS TO ASK

- Are production databases backed up? When was a restore last tested?
- If the backend went down right now, what is the plan, and who runs it?
- What was our last serious incident? How long did it last, and what changed afterward?
- If our main developer or vendor disappeared, could someone else build and ship the app?

EVIDENCE TO REQUEST

- ☐ Backup settings and the date of the last test restore
- ☐ A written review of a real past incident
- ☐ A short runbook for the most likely failures
- ☐ Proof the app builds from a fresh copy of the code on someone else's machine

WARNING SIGNS

- Backups exist but have never been restored.
- Incidents end with "it's fixed" and nothing written down.
- Only one person knows how to deploy the backend.

Your read

- ☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

AREA 07 Access to essential accounts

If you don't control the accounts, you don't fully control the business. Every account your app depends on should belong to the company, have more than one person who can get in, and not be tied to anyone who has left.

QUESTIONS TO ASK

- Is the Apple Developer account in the company's name? Who holds the Account Holder role?
- Who owns the Google Play Console account?
- Where are signing keys and certificates stored, and who can reach them?
- If a developer left on bad terms tomorrow, what could they lock us out of?

EVIDENCE TO REQUEST

- ☐ The account inventory on page 6, filled in together
- ☐ The admin list for each account
- ☐ Where two-factor codes and recovery codes go

WARNING SIGNS

- A developer or agency owns the Apple or Google account.
- Two-factor codes go to a former employee's phone.
- Nobody knows where the Android upload key is kept.

Your read

- ☐ Solid ☐ Shaky ☐ Unknown

Notes and evidence still missing

ACCOUNT INVENTORY

ACCOUNT	COMPANY OWNED?	ADMINS (NAMES)	TWO-FACTOR AND RECOVERY GO TO	CHECKED
Apple Developer account (Account Holder)				
Google Play Console				
Code repository (GitHub or similar)				
Subscription platform (RevenueCat, etc.)				
Backend and cloud hosting				
Domain registrar and DNS				
Analytics and crash reporting				
Email and push notification services				
Signing keys and certificates				
Password manager				

YOUR READS AT A GLANCE

	01	02	03	04	05	06	07
SOLID	☐	☐	☐	☐	☐	☐	☐
SHAKY	☐	☐	☐	☐	☐	☐	☐
UNKNOWN	☐	☐	☐	☐	☐	☐	☐

01 Ownership · 02 Visibility · 03 Journeys · 04 Releases · 05 Maintenance · 06 Recovery · 07 Access

CHOOSING WHAT TO FIX FIRST

- Pick no more than two. Two things done this month beat seven things started.
- Start with any Unknown in Access or Recovery. Those are the hardest gaps to close in the middle of a crisis.
- Every action gets one owner, by name, and a date.
- Write down what “done” looks like, so you can check it without reading code.

NEXT ACTIONS

AREA	NEXT ACTION	OWNER (ONE NAME)	DONE LOOKS LIKE	DUE
Example: 07 Access	Move the Apple Developer account into the company's name	Founder, with the agency	Company is Account Holder. Agency keeps an Admin role.	30 days

More Unknowns than you'd like? Vermillion takes ownership of engineering for live mobile apps, so founders can get back to customers, partnerships, and growth. A 15-minute call is enough to see where to start.

vermillionsky.com/contact