The Technical Decision Framework

A Non-Technical Founder's Guide to Evaluating Tech Decisions, Teams, and Architecture

By **Kuanysh Bay** · 13+ years building production systems
prodmake.com · kuan.sh

Why This Matters

42% of startups fail because they don't solve a real market need. But here's what the statistics don't tell you: many of those failures were sealed not by the idea, but by technical decisions made in the first six months.

For non-technical founders, the tech stack feels like a black box—something the dev team "handles." But early choices often seem fine, until months later, when you realize you've built something fragile, expensive to maintain, and nearly impossible to scale.

This framework gives you the questions to ask, the red flags to spot, and the mental models to evaluate technical decisions—without writing a single line of code.

Part I: The Five Questions

Before approving any significant technical decision, ask these five questions. The answers reveal more than any technical explanation.

> *1. "What happens if this breaks at 3 AM?"*
>
> **Testing:** Operational maturity and risk awareness.
>
> **Good answer:** Specific incident response plan. Monitoring alerts. Named person on-call. Automated recovery.
>
> **Red flag:** Vague reassurances. "It won't break." No mention of monitoring.
>
> **Why:** Every system breaks. The question is whether your team has thought about failure modes before they happen.

2. "If you left tomorrow, how long until someone else could maintain this?"

Testing: Knowledge distribution and documentation.

Good answer: "A few days. Here's our documentation. Here's the architecture diagram."

Red flag: Nervous laughter. "I'm the only one who really knows how this works."

Why: Key-person dependency is a silent killer of startups.

3. "What did we decide NOT to do, and why?"

Testing: Decision-making rigor and trade-off awareness.

Good answer: Clear articulation of alternatives. Specific reasons each was rejected.

Red flag: Can't name alternatives. "This is just how it's done."

Why: If your team can't articulate what they traded away, they defaulted to the familiar.

4. "What's the most likely way this fails in 12 months?"

Testing: Long-term thinking and technical debt awareness.

Good answer: Specific scaling limits. Known constraints. Plan to address them.

Red flag: "It won't fail." No awareness of limitations.

Why: Technical debt compounds like financial debt.

5. "How would a senior engineer from a top company critique this?"

Testing: External perspective and humility.

Good answer: Thoughtful self-critique. Awareness of where they cut corners.

Red flag: Defensiveness. "They'd love it."

Why: The best engineers are their own harshest critics.

Part II: The Red Flags Checklist

Warning signs that experienced CTOs recognize immediately. *If you see three or more, get an outside review.*

Architecture

- ☐ **No architecture diagram exists** — If they can't draw it, they don't understand it.
- ☐ **"We'll refactor later"** — Later never comes.
- ☐ **Single database for everything** — Bottleneck at scale.
- ☐ **No staging environment** — Testing in production.
- ☐ **Manual deployments** — Humans make errors.

Team

- ☐ **One person knows everything** — Existential risk.
- ☐ **No code review process** — Every change goes straight to production.
- ☐ **High developer turnover** — People leave bad codebases.
- ☐ **Resistance to outside review** — "You wouldn't understand."
- ☐ **No on-call rotation** — Who fixes things when they break?

Process

- ☐ **No written documentation** — Tribal knowledge disappears.
- ☐ **Estimates always wrong** — Poor understanding.
- ☐ **Constant firefighting** — Reactive, never proactive.
- ☐ **"Just one more feature"** — Weak technical leadership.
- ☐ **No metrics or monitoring** — Flying blind.

Part III: Vendor Evaluation Matrix

Use this when evaluating agencies, contractors, or hires.

Factor	Weight	Score	Notes
Relevant experience	20%		Built something similar?
Communication clarity	20%		Can explain simply?
Reference quality	15%		What clients say?
Process maturity	15%		Documented processes?
Team stability	10%		Key people tenure?
Technical depth	10%		Answer "why"?
Cultural fit	10%		Values aligned?

Scoring: 5 = exceptional · 4 = strong · 3 = adequate · 2 = concerning · 1 = disqualifying

Interpretation:

- **4.0+** — Strong candidate
- **3.5–4.0** — Proceed with clear agreements
- **3.0–3.5** — Significant risk
- **Below 3.0** — Do not proceed

Part IV: The Four Stack Traps

Common patterns that lead to costly rebuilds.

TRAP 01

Choosing Familiarity Over Fit

The team picks technologies they know, even when suboptimal. Building a real-time app with tools for static websites.

Avoid: Ask "What would you choose if skill constraints didn't exist?" The gap reveals risk.

TRAP 02

Chasing Trends

Adopting the latest framework because it's popular, not proven. Bleeding-edge tech with no mature ecosystem.

Avoid: Ask "How many production apps use this at scale?" Require examples, not hype.

TRAP 03

Overengineering (or Under-)

Enterprise architecture for an MVP. Or a prototype that can never scale. Microservices for a team of three.

Avoid: Ask "What's the simplest thing for the next 18 months?"

TRAP 04

Letting Budget Drive Architecture

Cheapest option without considering long-term costs. Low-code that works until it doesn't.

Avoid: Calculate rebuild cost. Compare to "savings." The math rarely favors cheap.

Part V: When to Bring in Help

1. **Before raising funding** — Investors require due diligence. Don't be surprised.
2. **Before a major pivot** — New direction may need new architecture.
3. **When velocity drops** — Features take 3× longer. Something's wrong.
4. **Before acquisition** — Due diligence determines valuation.
5. **When your gut says something's wrong** — Founder intuition is often right.

Quick Reference

Before Every Major Decision

- ☐ Can the team explain it simply?
- ☐ Alternatives documented?
- ☐ Rollback plan if it fails?
- ☐ More than one person understands it?
- ☐ Way to measure success?

Monthly Health Checks

- ☐ Production incidents this month?
- ☐ Average time-to-deploy?
- ☐ Estimates improving?
- ☐ New single points of failure?
- ☐ Tech debt trend?

Quarterly Strategic Review

- ☐ Can architecture support 10× growth?
- ☐ Technologies with active communities?
- ☐ Runbooks for critical operations?
- ☐ Bus factor greater than one?
- ☐ What would an auditor find?

About the Author

Kuanysh (Kuan) Bay has spent 13+ years shipping production systems at scale. Infrastructure he's built serves 22M+ users. Systems he's contributed to have been part of $B+ exits.

He works as a fractional CTO and technical advisor, helping non-technical founders make better technical decisions.

prodmake.com · kuan.sh · Book a call