

FREE RESOURCE

The Manager's Architecture Cheat Sheet

Twenty terms you will hear, what each one costs you, and the question that turns a statement into a decision.

Drawn from T3 · Architecture for Managers

You do not need to know how any of this is built. You need to know what it costs, how reversible it is, and what to ask next. That is the entire list.

Print it. Keep it next to you in the meeting where someone says a word you have been nodding at for two years.

TERM	WHAT IT ACTUALLY MEANS	WHAT IT COSTS YOU	ASK THIS
Monolith	One deployable unit. All the code ships together.	Cheap to build and easy to reason about. Every change ships with every other change, so releases get slower as the team grows.	<i>“What is actually slow — the build, the tests, or the agreement between teams?”</i>
Modular monolith	One deployable unit with hard internal boundaries.	Most of the benefit of microservices without the operational tax. Usually the right answer, and rarely the exciting one.	<i>“What would have to be true before we split this apart?”</i>
Microservices	Many independently deployable units talking over a network.	Independent releases, and a permanent tax: networking, tracing, versioning, and much harder debugging. Pays off with many teams, not many features.	<i>“How many teams are blocked on each other today?”</i>
API contract	The promise one system makes about what it accepts and returns.	Once someone builds on it, changing it becomes their problem too. Cheap to write, expensive to break.	<i>“Who depends on this today, and how would we find out?”</i>
Breaking change	A change that makes existing callers fail.	Coordination with everyone downstream, or a new version. The cost is rarely the code; it is the calendar.	<i>“Can we add rather than change?”</i>
Deprecation	Announcing something will stop working, then removing it.	A long tail of stragglers. Deprecation without a removal date is just a note.	<i>“What is the removal date, and who tells the users?”</i>
Coupling	How much one part must know about another to work.	Tight coupling makes every change ripple. It is why a small feature sometimes touches nine files.	<i>“What else has to change when this changes?”</i>

TERM	WHAT IT ACTUALLY MEANS	WHAT IT COSTS YOU	ASK THIS
Technical debt	A shortcut, deliberate or not, that makes future work slower.	Interest, paid on every subsequent change. Not a moral failing — an investment position with a running cost.	<i>“What is this costing us per sprint, and what would fixing it buy back?”</i>
Refactor	Changing internal structure without changing behaviour.	Real time, zero visible output. Legitimate and unsellable at once, which is exactly why it needs a number attached.	<i>“What gets faster afterwards, and by how much?”</i>
Rewrite	Rebuilding something from scratch.	Almost always longer than estimated — and the old system keeps changing while you do it. The most expensive word on this page.	<i>“What does the current system do that we have all forgotten about?”</i>
Migration	Moving data or traffic from one system to another.	Runs alongside everything else, produces nothing visible, and cannot be safely left half-finished.	<i>“What is the rollback, and how long can both run side by side?”</i>
Schema change	Changing the shape of stored data.	On a large table this is not a code change, it is an operation with downtime risk. “Just add a field” is sometimes a sprint.	<i>“How big is the table, and does this need a backfill?”</i>
Index	A lookup structure that makes reads fast.	Faster reads, slower writes, more storage. Adding one is easy; knowing which one is the skill.	<i>“Which query is slow, and how slow?”</i>
Cache	A copy of an answer kept somewhere closer.	Speed, in exchange for the possibility of being wrong. Every cache is a decision about how stale is acceptable.	<i>“How stale can this be before someone complains?”</i>
Queue / async	Work handed off to be done shortly, not now.	Resilience and throughput, in exchange for “done” no longer meaning “finished”.	<i>“What does the user see between the request and the result?”</i>
Eventual consistency	Different parts of the system briefly disagree.	Scale and availability, in exchange for a support ticket reading “I paid and it says I didn’t”.	<i>“How long is ‘eventually’, and who sees the gap?”</i>
Idempotent	Doing it twice has the same effect as doing it once.	The difference between a retry being safe and a customer being charged twice.	<i>“What happens if this runs twice?”</i>
Feature flag	A switch that changes behaviour without a release.	Release decoupled from deploy — and a growing pile of flags nobody ever removes.	<i>“Who turns it off, and when does it get deleted?”</i>
Environment	A separate running copy: local, staging, production.	Confidence, and the cost of keeping them alike. Most “it worked in staging” problems are really “staging is not like production”.	<i>“How is staging different from production?”</i>

TERM	WHAT IT ACTUALLY MEANS	WHAT IT COSTS YOU	ASK THIS
SLO / error budget	The agreed reliability level, and how much failure is allowed.	A shared language for “is this good enough?” — and a way to say no to features without saying no.	<i>“What is our error budget, and have we spent it?”</i>

Four phrases that usually mean something is being avoided

None of these are lies. All four are places where a cost exists and has not yet been said out loud. The question in each row is the cheapest way to surface it without an argument.

WHAT YOU HEAR	WHAT IT USUALLY MEANS	ASK THIS
"It's technically possible."	Everything is technically possible. This means the cost is high and I do not want to name it yet.	<i>"What would it cost — in weeks, and in what we would give up?"</i>
"We need to look into it."	There is no owner and no date. This will resurface unchanged in three weeks.	<i>"Who is looking, and when do we talk again?"</i>
"It's a quick win."	It is almost never a quick win. Someone is compressing an unknown into an adjective.	<i>"What is the smallest version we could ship on Friday, and what breaks if we do?"</i>
"We should just rewrite it."	A wish, not a plan. Usually a symptom of pain somewhere specific that has never been named.	<i>"What does the current system do that we would have to rebuild before anyone noticed?"</i>

THE ONLY RULE UNDERNEATH THIS PAGE

Nothing here asks you to understand the implementation. Every question does the same thing: it turns a technical statement into a cost, a date or an owner — and those are three things you are qualified to decide about.

The Bridge is a training academy for product, engineering and data leaders — workshops, programmes, mentoring and advisory on the decisions the three functions have to make together. Taught live by a practising CTO. Everything here is drawn from the catalogue and published openly. **thebridge.academy**