The Rewrite Decision

Every engineering team eventually asks for a rewrite. Most of them should not get one, and the ones that should are usually asking for the wrong reasons.

A rewrite isn't a technical decision. It's a bet that you can rebuild years of accumulated behaviour, including the parts nobody documented, while the old system keeps earning and the market doesn't move. Teams lose that bet more often than they win it, and they lose it slowly enough that nobody can point at the day it went wrong.

This is the framework I use when someone asks me to bless one.

> Before anything else. Write down what will be true after the rewrite that isn't true now, in terms a customer would notice. If the list is empty, or every item is about how the code feels to work in, you don't have a rewrite case. You have a morale problem, and a rewrite is an expensive way to treat it.

Score the real motivation

Tick what is true. These are the reasons that hold up.

OK	CON	N/A	
☐	☐	☐	A named business capability is impossible in the current design, not merely awkward
☐	☐	☐	The cost of the next twelve months of change exceeds the cost of replacement, with numbers
☐	☐	☐	A hard external deadline exists that the current system can't meet: a regulation, a platform deprecation, a contract
☐	☐	☐	The system can't meet a load you're already contractually committed to
☐	☐	☐	A dependency at the core is dead, with no maintainer and no fork
☐	☐	☐	Security can't be brought to an acceptable level in place

Now tick what is also true. These are the reasons that don't hold up, and they are the ones usually driving the request.

OK	CON	N/A	
☐	☐	☐	The team dislikes the language or framework
☐	☐	☐	A new hire has done it differently somewhere else
☐	☐	☐	The code is ugly but works
☐	☐	☐	Nobody understands it, and reading it feels slower than rewriting it
			—This is comprehension debt. A rewrite converts it into a much larger version of the same problem, in a codebase with no users yet.
☐	☐	☐	The original authors have left
☐	☐	☐	It would be a good project for retention
☐	☐	☐	A vendor or a conference talk suggested it

If your second list is longer than your first, stop. The honest answer is a programme of refactoring with a budget and a deadline, which is harder to sell internally and works far more often.

What a rewrite actually costs

OK	CON	N/A	
☐	☐	☐	You have written down the full behaviour of the current system, including the undocumented parts
☐	☐	☐	You know which of that behaviour is load-bearing and which is accident
☐	☐	☐	You have an estimate, and you have doubled it —Not pessimism. The estimate covers the behaviour you know about.
☐	☐	☐	Somebody owns keeping the old system alive throughout, and that's a real allocation
☐	☐	☐	You have decided what happens to feature requests during the rewrite
☐	☐	☐	You have decided what you will tell customers when the roadmap stops
☐	☐	☐	You know how long the organisation can tolerate two systems
☐	☐	☐	You have budget for the migration of data, not just of code

Conditions where rewrites succeed

The successful ones I have seen share most of these. Count them.

OK	CON	N/A	
☐	☐	☐	The scope is one bounded component, not the system
☐	☐	☐	The old system keeps running and is retired incrementally
☐	☐	☐	There's a working strangler path: new code takes real traffic early
☐	☐	☐	Success is measured in traffic migrated, not in code written
☐	☐	☐	The team that will operate it is the team building it
☐	☐	☐	Someone with authority can stop it at any checkpoint
☐	☐	☐	The rewrite has a deadline that costs something to miss
☐	☐	☐	Requirements were rewritten first, in plain language, by people who use the system

Conditions that predict failure

OK	CON	N/A	
☐	☐	☐	Big bang cutover, planned for a single weekend
☐	☐	☐	No traffic on the new system until it's finished
☐	☐	☐	The old system is frozen while the new one is built
☐	☐	☐	The rewrite is also a change of language, framework, database and architecture at once
☐	☐	☐	The people who understood the original are not involved
☐	☐	☐	Progress is reported as percentage complete
☐	☐	☐	There's no agreed definition of done that a non-engineer could verify

The alternatives you should price first

OK	CON	N/A	
☐	☐	☐	Strangle it: route new work to a new component, migrate one path at a time
☐	☐	☐	Carve one bounded piece out and leave the rest
☐	☐	☐	Buy the capability instead of building it
☐	☐	☐	Refactor behind the existing interface, with tests as the safety net
☐	☐	☐	Improve the tests first, then decide again in a quarter
			—Often the fastest route. A system with tests stops feeling like it needs replacing.
☐	☐	☐	Do nothing, and revisit when a business reason appears

Make the call

OK	CON	N/A	
☐	☐	☐	Which reason from the first list is driving this, in one sentence?
☐	☐	☐	What does the customer get, and when?
☐	☐	☐	What is the smallest version that proves the approach?
☐	☐	☐	What is the checkpoint at which you would stop?
☐	☐	☐	Who decides to stop, and do they have the standing to do it?
☐	☐	☐	If this takes twice as long as planned, is the company still fine?

If you can’t answer the last one with yes, the rewrite isn’t a technical risk. It’s a company risk wearing a technical costume.