unify

unifygtm.com

To vibe or not to vibe

The GTM Guide to Building vs. Buying When Everyone Can Code

Build vs. buy is not a new question. Every ops leader, every founder, every revenue team has faced it at some point. What is new is that AI has made building accessible to everyone. Tools like Claude Code have removed the technical barrier entirely, and "did you try to build it first?" has become the default in almost every budget and tooling conversation.

The question is no longer whether you can build. You almost certainly can.

The question is:
Which things are worth building?

This guide is a framework for making that call.

Why Building Is Easier Than Ever

AI has removed the technical barrier to building internal tools. Platforms like Claude Code, Codex, Replit, and Cursor now let non-technical operators ship automations on their first try. A year ago, this meant hiring an engineer or learning to code. Today, you just describe what you want and get a working result.

70%+ of new applications are now built using low-code or no-code tools
according to Gartner

GTM teams are already taking advantage:

- **RevOps leads** are building agents that run closed-lost analysis across multiple data sources.
- **Growth teams** are spinning up account research workflows in an afternoon.
- **Sellers** are creating custom dashboards and Slack automations that would have sat in a backlog for months.

For contained, low-lift use cases, building is faster and cheaper than it has ever been, and the teams that ignore this are leaving speed and customization on the table.

But the fact that building got easier doesn't mean the decision got simpler. The costs that burn teams have never been about the build itself.

The Hidden Costs of Building

The hardest part of the build vs. buy decision is that the costs you can plan for are only a fraction of the costs you actually pay. The teams that get burned by building almost always struggle with the same handful of things that never show up in the original estimate.

Integration issues

When something breaks across four stitched-together tools with no shared observability, there is no support team to call because you are the support team.

Compounding maintenance

Every new team, process, tool, or field update requires rework across the custom stack.

The maturity gap

Platforms you would consider buying have already absorbed years of edge cases you haven't encountered yet.

Single-person risk

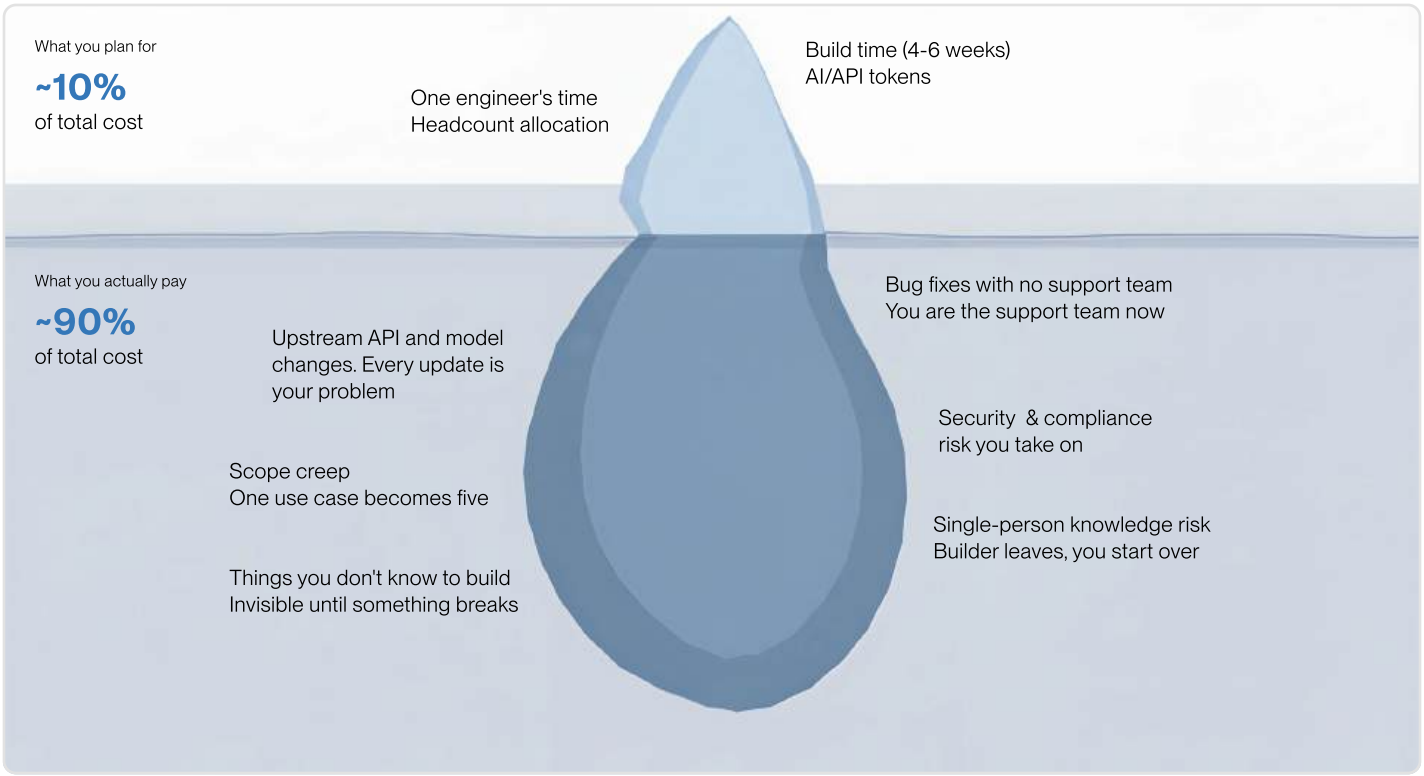
The person who built the thing is usually the only person who understands it, and when they leave, it creates internal fires and thrash that are difficult to stop.

Scope creep

You built for one use case, the team wants multi-channel, then A/B testing, then reporting. Soon your tool looks nothing like what you started with.

Compliance exposure

Data governance, privacy regulations, and security requirements get more complex as you scale, and a homegrown stack puts the burden of staying compliant on you.



The Compounding Cost of “Good Enough”

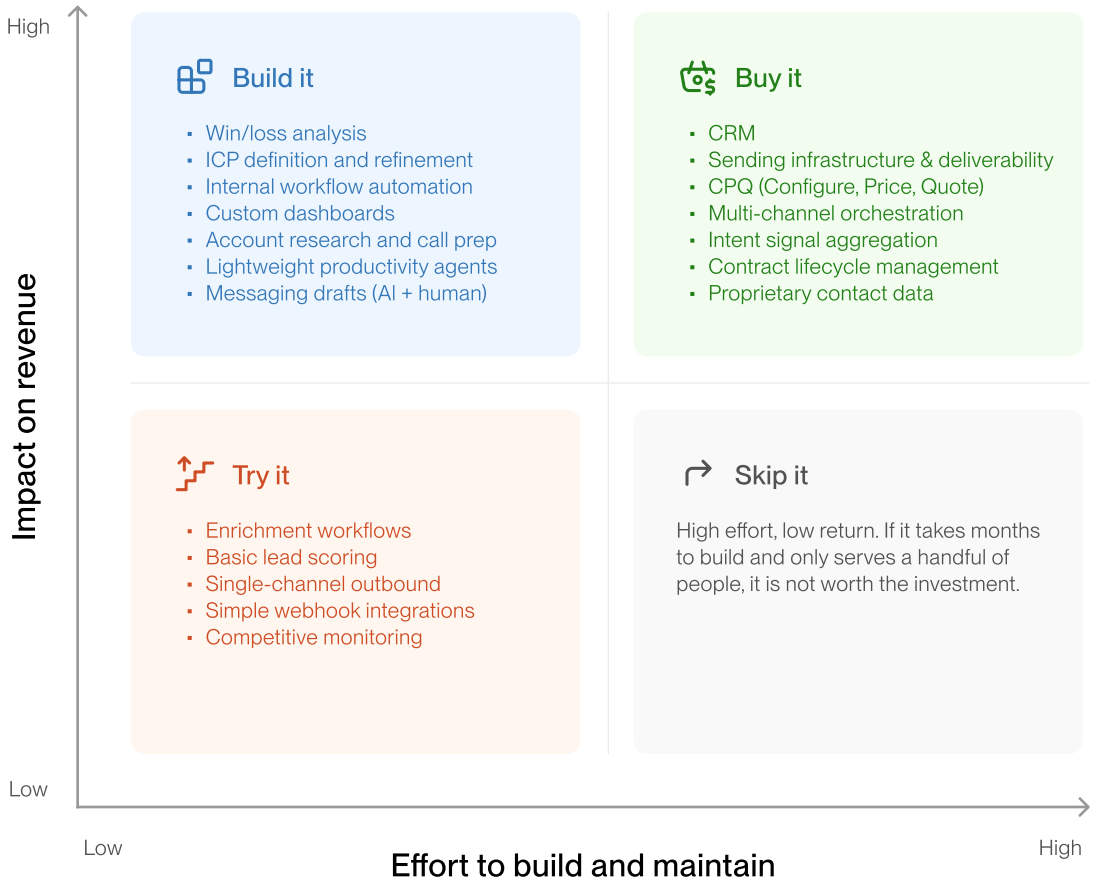
The biggest risk in build vs. buy isn’t a single bad decision, but the long-term compounding effects of a cobbled-together stack that works just well enough that nobody questions it. When each tool operates in isolation, data doesn't connect and nobody can measure full-funnel impact. Every month, the gap between what you have and what you could have gets bigger.

> **If vibe-coded tools could truly replace mature platforms, those platforms would be dead already. They are not.**

The Build It, Try It, Buy It Framework for GTM Teams

The answer to build vs. buy is almost never all one or the other. Some things are worth building because the lift is low and the payoff is immediate. Some things are worth buying because the cost of getting them wrong is too high. And a whole category in between depends on your stage, your resources, and how much risk you can absorb.

Think of every tool and workflow in your GTM stack as falling into one of four buckets based on two variables: the effort required to build and maintain it, and the impact it has on revenue.



Build it

These are the use cases where AI and vibe coding shine. The lift is low enough that you can ship something useful in days, the scope is contained, and the output is customized to your exact motion.

Try it

These use cases fall into a gray area. They can work if you have the right person, the right scope, and realistic expectations about maintenance. But they can also spiral if you don't. Go in with your eyes open.

Buy it

These are use cases where the maintenance burden, compliance exposure, data requirements, make building a bad bet for almost every team. The vendors in these spaces are sitting on years of product maturity and proprietary data you can’t replicate.

Skip it

These are projects that feel productive but aren't. The effort to build and maintain them is significant, and the output only serves a small number of people or a narrow use case. If it takes months to ship and doesn't move pipeline, your time is better spent somewhere else.

The 8 Questions to Ask Before You Build Anything

The framework gives you general categories, but actually categorizing a project can be challenging. Ask yourself these questions before you commit to anything to better understand the risks and opportunities. Use the scorecard on the next page as a reference.

1. **What is the impact on revenue?**
   Is this going to directly generate, accelerate, or recover revenue, or is it a nice-to-have that sits at the edges of your motion?

2. **Who is this for?**
   One person, one team, or the whole org? The cost of building is the same regardless, but the return scales with the number of people who benefit from it.

3. **How much time and effort will it take to build?**
   Days, weeks, or months? Projects that ship in days carry a very different risk profile than projects that require months of iteration before they deliver any value.

4. **How much maintenance will it need?**
   Every tool you build is a tool you now own. If it requires ongoing rework every time a play changes, a field updates, or an API moves, factor that into the real cost.

5. **What is the risk if it breaks or you get it wrong?**
   A broken internal dashboard is an inconvenience. A broken sending infrastructure is burned domains and dead pipeline. The blast radius determines whether this is something you can afford to own.

6. **How much would buying cost?**
   Put the platform price on the table and compare it honestly against the fully loaded cost of building, maintaining, and iterating over time.

7. **What is the opportunity cost of building?**
   The person building this is not doing something else. What is that something else, and is it more or less valuable than the thing they are building?

8. **How much control do you actually need?**
   The more control you need over a workflow, the stronger the case for building it yourself. If it needs to be fully customized to your motion, owning it makes sense. If it just needs to work, buying gets you there faster and frees up your time for other projects.

	1 point	2 points	3 points
Pipeline impact	Nice-to-have	Helps, not critical	Revenue-critical
Who is it for	Just me	My team	Whole org
Time to build	Days	Weeks	Months
Maintenance load	Minimal	Periodic rework	Constant upkeep
Risk if it breaks	Inconvenience	Disrupts workflow	Pipeline at risk
Cost to buy instead	Expensive	Comparable	Cheaper to buy
Opportunity cost	Builder has capacity	Some tradeoffs	Needed elsewhere
Control needed	Must be custom	Some flexibility	Just needs to work

Tally	___ × 1 = ___	___ × 2 = ___	___ × 3 = ___
Total score			___ / 24

8 – 13	14 – 18	19 – 24
Build it	Try it	Buy it

So, To Vibe or Not to Vibe?

The question isn't whether you can build. You almost certainly can. The question is whether the thing you're building is the highest and best use of the time, the talent, and the focus you are putting into it. For some things the answer is yes, and you should build with confidence. For others the answer is no, and the faster you arrive at that conclusion the more time you save for the work that actually increases revenue.

Use the framework and scorecard to get a better sense of where your project falls on the spectrum. The most important thing is to be honest with yourself about what comes post-launch.

unify

unifygtm.com

Make outbound your best pipeline channel

If you're looking to level-up your outbound program for the AI era, come chat with the Unify team to see what's possible.

Talk to Our Team