ENTERPRISE DNA

A DECISION GUIDE FROM ENTERPRISE DNA

The GitHub Copilot Decision Guide

Pricing, plans, and when to switch, after the June 2026 move to AI Credits. What now meters, what a seat really costs, and how to keep the bill under control.

FRONT MATTER

A note from Sam

If you searched for GitHub Copilot pricing this year and came away unsure what a seat actually costs now, you are not imagining it. The ground moved on June 1, 2026.

On that day GitHub retired flat per-seat access and moved every Copilot plan to token-metered billing, priced in AI Credits. Code completions stayed free. Almost everything else, chat, agent mode, code review, now draws down a monthly pool. The official announcement drew nearly a thousand downvotes and hundreds of comments, which tells you how it landed.

This guide is the neutral read we could not find anywhere else. What changed, what each plan really costs, why one agent run can spend more than a week of chat, how to govern the bill, and an honest look at the alternatives. We are not selling you Copilot, Claude Code, or Cursor. We use several of these tools ourselves and we will tell you where each one wins.

We run our own company on a command center of agents, with a person on the gate. So we feel a metered agent bill the same way you do. Read this as a field guide, then make the call that fits your team.

Sam McKay

Founder, Enterprise DNA

CONTENTS

What is inside

1 **What changed.** The June 2026 move to AI Credits, and what still stays free.

2 **The plans, real costs.** Free to Enterprise, credits, models, and pooling.

3 **The agentic math.** Why an agent session burns fast, and how to estimate your team's real spend.

4 **Budgeting and governance.** The control levers, and the visibility gap.

5 **The alternatives, honestly.** Copilot, Claude Code, and Cursor for a team.

6 **The decision.** Stay, downgrade, or switch, by team profile.

7 **The checklist.** One page of cost control you can hand to the team.

8 **Where Enterprise DNA fits.** How we help, and how to start.

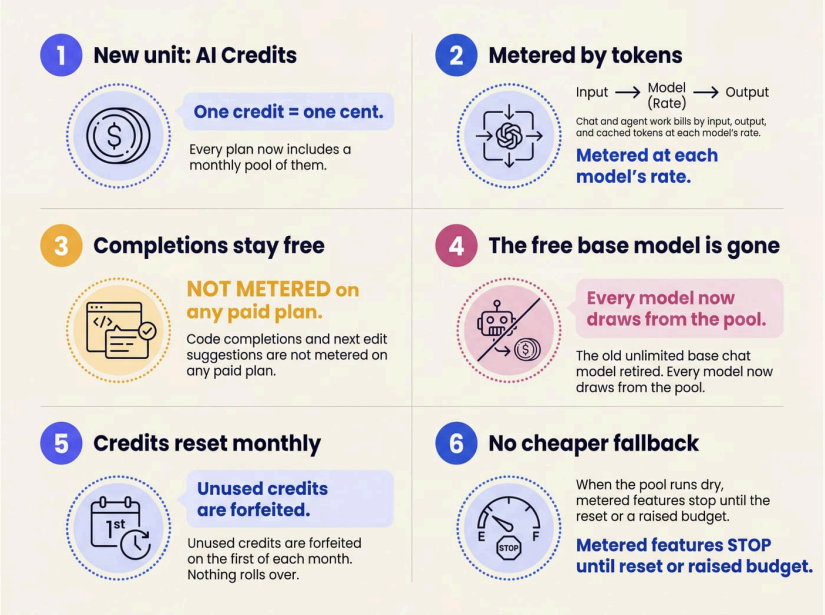
PART 1 · WHAT CHANGED

The day the meter switched on

For years, GitHub Copilot was a flat deal. You paid a monthly seat fee and used chat and agents fairly freely. On June 1, 2026, that ended.

Copilot moved to usage-based billing. Your seat fee now buys a pool of AI Credits, where one credit is one cent. Work is metered by the tokens it consumes, input, output, and cached, at each model's rate. The seat price did not change. What the seat buys did.

> *The seat still costs the same. What the seat buys is now metered.*



The good news is narrow but real. Code completions and next edit suggestions, the inline autocomplete most people think of as Copilot, stay unlimited on every paid plan. If your team uses Copilot only that way, the change barely touches you. The rest of this part is about everything else.

What stays free, what now meters

The line between free and metered is the single most important thing to understand about the new model. It is a sharp line, and most of the product sits on the metered side.

Free on every paid plan. Code completions and next edit suggestions are not billed in credits. They keep working even when your credit pool is empty.

Metered from the first token. Chat, agent mode, code review, the Copilot CLI, and the cloud coding agent all draw credits. The old unlimited base chat model is gone, and so is the automatic fallback to a cheaper model when you run out. When the pool is dry, metered features stop until the monthly reset or a raised budget.

So the question is not whether you use Copilot. It is how much of your use is agentic. That is where the bill now lives.

PART 2 · THE PLANS, REAL COSTS

The plans at a glance

The prices held through the change. What each plan now includes, in credits, is the number that actually decides your bill.

Read a plan by its credits, not its price. Each plan bundles a monthly credit pool, and every plan except Free keeps completions unlimited on top. The individual tiers run from Free through Pro and Pro Plus to Max. The team tiers, Business and Enterprise, pool credits across seats and add admin controls. The next two pages walk each side.

All prices in US dollars per month. Team prices are per user. Figures verified against GitHub's pricing pages, July 2026, and can move.

Individuals: Free to Max

Four individual tiers, each buying a larger credit pool. Completions stay free on all of the paid ones.

Free, $0
Completions plus a small monthly allowance of chat and agent on a limited model set. No credit pool.

Pro, $10
About ten dollars of credits, plus an adjustable flex top-up. The everyday paid tier.

Pro Plus, $39
A larger pool and access to premium models such as Claude Opus and the top GPT tier.

Max, $100
The heavy individual tier, built for near-constant agent use.

The flex catch

Pro and Pro Plus carry a "flex allotment" of extra credits on top of the base. It is generous today, but GitHub can adjust it as model prices move. Plan against the base you are contractually owed, and treat the flex as a bonus that could shrink, not a floor.

Unused credits are forfeited. The pool resets to its full amount on the first of each month. Nothing rolls over, so a light month does not bank credit for a heavy one.

Teams: Business and Enterprise

The team tiers change one thing that matters more than the price. They pool credits across every seat.

Business, $19 per user
Pooled credits across the team, admin budgets, and org policy over which features run.

Enterprise, $39 per user
Pooled credits, cost-center budgets, and the full set of org and security controls.

Pooling is the point

Credits pool at the account level. A developer who runs agents all day can draw on the credits a lighter colleague never touches. GitHub calls this killing stranded capacity, and for a mixed team it is the single biggest reason the team tiers price out better than the same number of individual seats.

A promo is running. Through around September 1, 2026, Business and Enterprise seats carry extra promotional credits on top of the base. Budget for the base allowance, because that is what remains once the promo ends. Then set the controls in Part 4 before anyone starts a long agent run.

PART 3 · THE AGENTIC MATH

Why an agent run burns fast

A chat message and an agent task look similar on screen. On the meter they are worlds apart. Understanding why is how you avoid the surprise bill.

An agent task is not one model call. The agent plans, reads files for context, drafts a change, runs tests, and retries on failure. Each step is a call, and the context it has gathered rides along on every later request in the loop. A chat turn might be a few thousand tokens. One agent session can top half a million.

The range is wide, and the top end is steep. A moderate agent task might cost a dollar or a few. A complex, multi-file run on a premium model can reach three to four thousand credits, thirty to forty dollars, in a single session. Developers have reported half a monthly allowance gone in one request. Point an agent at your whole repository when three files matter, and you pay to read the whole repository.

Estimate your team's real burn

You do not have to guess. GitHub's usage data lets you measure a baseline and project it, before the bill arrives rather than after.

A method a team lead can run

Find it: Billing and licensing, then AI usage. Filter by user, model, and cost center.

Baseline a week: pull one representative week and separate free completions from metered spend.

Project it: multiply the metered week out to a month, then split by developer and model.

Find the heavy few: a small number of agent power users usually drive most of the spend.

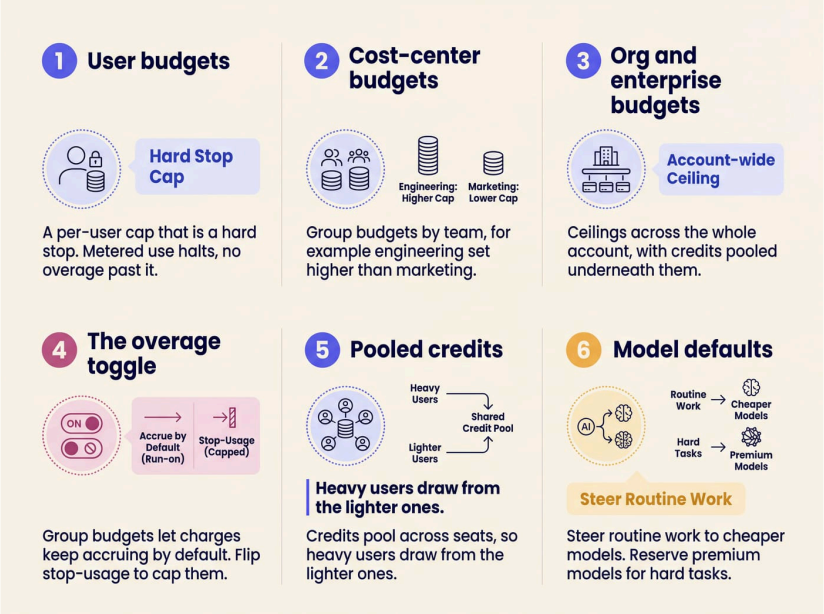
Set model defaults deliberately. Route routine work to a cheaper model and keep the premium models for the genuinely hard tasks. A small model can cost a fraction of a premium one for the same token count, so the default you pick quietly sets your monthly number.

One honest caveat. The dashboard shows history, and GitHub does not publish how fresh the numbers are. Treat it as a rear-view mirror, not a live speedometer. That gap is the subject of the next part.

PART 4 · BUDGETING AND GOVERNANCE

The control levers

The good news for a team lead is that the tools to cap spend do exist. The catch is that the most important one is off by default.



Budgets sit at four levels. You can cap spend per user, per cost center, per organization, and across the enterprise. User-level budgets are a hard stop. Once a user hits the cap, metered features halt, with no overage past it, though their completions keep working.

> *Group overage is on by default. It is on until you turn it off.*

The lever to set first. At the group level, charges keep accruing past the budget unless you flip the stop-usage toggle. Turn it on. Then pool your credits so heavy users draw from light ones, and set model defaults so routine work does not reach for a premium model.

The visibility gap

The hardest part of the new model is not the price. It is not being able to see the spend as it happens.

There is no cost estimate before you press send. A prompt can quietly consume a large slice of the monthly pool, and nothing warns you first. The dashboard reports after the fact, with unstated latency, so a team can discover an overrun a day after it happened. That is the gap between a cap that stops you and knowing what you are about to spend.

How to close it

Hard caps: set user budgets so the worst case is bounded, not open-ended.	**Named ownership:** track spend by developer and repo, so you know where it goes.
A weekly read: review usage every week, not once the invoice lands.	**Scope discipline:** a team habit of pointing agents at the files that matter.

Governance is a habit, not a setting. The controls bound the damage. What actually keeps the bill sane is a team that scopes its agent runs and reads its own usage. Someone has to own that, or the meter owns you.

PART 5 · THE ALTERNATIVES, HONESTLY

Three tools, three cost models

Copilot is not the only option, and the June change sent plenty of teams shopping. Here is the honest shape of the field. We sell none of these.

They bill in three different shapes. Copilot is a seat with a meter. Claude Code is a flat fee with a wall, you hit a usage limit and wait. Cursor is a pool with a spillover, included usage then overage. The shape matters as much as the sticker price, because it decides how predictable your month is.

The next two pages put the numbers side by side, then say plainly when each one is the right call.

The three cost models side by side

Same job, three ways of paying for it. The figures are current as of July 2026 and, like all of these, can move.

GitHub Copilot
Per seat plus metered AI Credits. Completions free, chat and agent billed by token. Business $19, Enterprise $39 per user.

Claude Code
One flat fee covers the apps and the coding agent. Pro $20, Max $100 or $200. No overage inside a tier, you hit a limit and wait.

Cursor
A usage pool then on-demand overage billed in arrears. Unlimited tab and a flat auto mode. Pro $20, Teams $40 per user.

Predictability, for the person signing off

Most predictable is the flat wall. Claude Code within a tier has no metered overage, so the number is fixed until you deliberately switch to pay-as-you-go. Copilot has a forecastable baseline, because completions stay free, plus variable agent spend on top. Cursor is the least predictable, since overage is billed after the fact, which is exactly what its own 2025 pricing backlash was about before it stabilized.

When each one wins

There is no single best tool. There is a best fit for how your team actually works. Match the tool to the pattern.

Copilot wins
For IDE-native teams already living in GitHub who want a choice of models and free completions as the cost anchor.

Claude Code wins
For agentic power users doing autonomous, multi-file, long-horizon work, where flat pricing tames a heavy habit.

Cursor wins
For in-editor flow and fast iteration, with a best-in-class autocomplete and a strong native editor.

Cost-first wins
Claude Code Pro at a flat $20, or Copilot with hard budgets, give the most predictable monthly number.

> *Match the tool to how the team works, not to the loudest review.*

Most teams land on a mix. Completions and light chat on Copilot, heavy agent work on a flat-fee tool, is a common and sensible split. The point is to decide on purpose, from your own usage data, rather than default into whatever you had before June.

PART 6 · THE DECISION

Stay, downgrade, or switch

Everything so far leads to one call. The honest answer is that it depends on your team, and it is usually a different call for different teams inside the same company.

Start from your usage, not from a preference. Split the free completions from the metered spend. If you are mostly completions, stay, and trim unused seats. If you are heavy on agents, price a flat-fee tool before you renew. If you need a predictable bill above all, a flat plan removes the surprise. If you are locked into GitHub for other reasons, stay, but set hard budgets and pool your credits.

Three team profiles, walked through

The tree is easier to read against real teams. Here are three common ones and the call each usually lands on.

The in-editor team

Mostly autocomplete and the odd chat. The June change barely touches them, because completions stayed free. **Stay, and downgrade.** Move light users to Pro or trim seats, and keep an eye on the few who drift into agent use.

The agentic power team

Autonomous, multi-file work all day. This is where metered credits bite hardest, and where a heavy month gets expensive fast. **Price the switch.** Put a flat-fee tool next to Copilot with hard caps and pooled credits, and compare a real month.

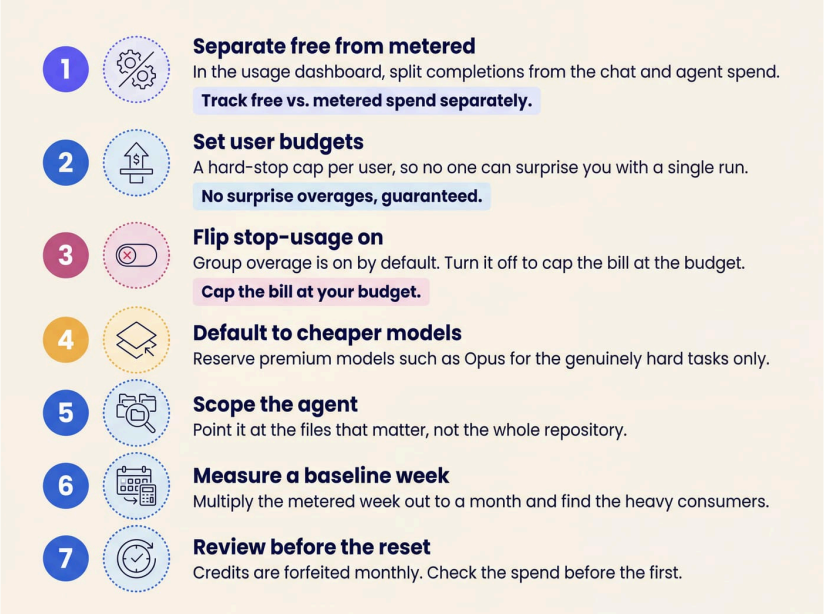
The data and analytics team

Exploratory, notebook-heavy, bursty. Either an agentic tool or an in-editor one can fit. **Match the pattern.** Choose on how they actually work, and govern whichever you pick with budgets and a weekly usage read.

PART 7 · THE CHECKLIST

The cost-control checklist

One page to keep the bill honest, whichever way you decide. This is the page to share with the team.



The order matters. Separate free from metered so you know your real exposure, set user budgets as hard stops, and flip stop-usage on before anyone starts a long run. Then default to cheaper models, scope every agent run, measure a baseline week, and review before the monthly reset takes your unused credits.

PART 8 · WHERE WE FIT

Where Enterprise DNA fits

> *We run our own company on exactly this, with a person on the gate.*

Enterprise DNA has spent more than a decade helping people do more with their tools and their data. AI coding assistants are one of many places the same lesson applies. The tool is only as good as the way you run it, and the bill is only as sane as the habits around it.

Tool sprawl is the real cost. A team on Copilot, Claude Code, Cursor, and three other subscriptions, each with its own meter, is exactly the situation an operating layer is built to tame. Someone runs the whole thing, watches the spend, and keeps a person on the gate.

Learn it
Training to get your team fluent with agents and disciplined about how they spend.

Build it
We help you set the budgets, the defaults, and the review habits on your own tools.

Run it
We operate the layer alongside your team, the way we run our own command center.

Start with your own usage

Open the AI usage dashboard, separate free from metered, and set one hard budget this week. If you want a second pair of eyes on the whole AI tool bill, we will help you read it.

Learn: enterprisedna.co	**Email:** sam.mckay@enterprisedna.co.nz
Talk: a thirty-minute call, no pressure.	**The decision:** yours.

ENTERPRISE DNA

A NOTE IN CLOSING

The seat price did not change. What the seat buys did. The teams that come out ahead are the ones who read their own usage, set a hard budget, and decide on purpose, rather than paying a meter they never watch.

FROM INSIDE THIS GUIDE.

Decide with the numbers in front of you.