A PRACTICAL GUIDE FOR TEAMS

Copilot at Work

Getting Real Value from Microsoft 365 AI

What the assistant is actually for

Why your content decides your results

The 30-day plan that ends in a decision

When Copilot isn't the answer, and what is

Rosemarie Withee

Author of six books on Microsoft 365

Copilot at Work

Copilot at Work

Getting Real Value from Microsoft 365 AI

Rosemarie Withee
Portal Integrators · Seattle

Copilot at Work: Getting Real Value from Microsoft 365 AI

Microsoft, Microsoft 365, Copilot, SharePoint, Teams, Outlook, Word, Excel,

A free guide from Portal Integrators
portalintegrators.com/writing

Contents

CHAPTER 1

What Copilot Is (and What It Isn't)

IN THIS CHAPTER

- The one-sentence definition of Copilot, and the four boundaries hiding in it
- The same task prompted badly and well, side by side
- The four things Copilot is not, including the expensive one
- Why confident wrong answers happen, and the norm that defuses them

The license shows up on a Tuesday.

Maybe you asked for it. Maybe your boss came back from a conference with it. Either way, there's now a colorful icon in your Microsoft apps, an invoice somewhere in accounting, and a quiet expectation hanging over the whole thing: this is supposed to change how we work.

Whether it actually does has surprisingly little to do with the software. The pattern goes like this. Licenses roll out to forty people the week after a vendor demo, and the demo showed an assistant that seemed to run the office. Three weeks in, the internal verdict is "it's useless": people asked it to process incoming client documents, watch a shared inbox, and update the tracking spreadsheet, and it did none of it. It was never going to. Those are automation jobs, and they bought an assistant. Then somebody resets the team on assistant-shaped tasks (catching up on threads, first drafts, finding answers in their own

files), and the same license goes from punchline to fixture in about a month. Nothing about the software changed. The expectations did.

That arc is the arc of this whole guide, so this chapter does the most valuable thing a guide like this can do. It tells you precisely what you bought.

One sentence, four boundaries

Here is the most useful sentence in this guide:

Copilot is an assistant for a person working inside a Microsoft document or conversation.

That sounds almost too obvious to write down. But every word of it is a boundary, and nearly every Copilot disappointment traces back to someone crossing one of these boundaries without noticing.

"An assistant." Copilot helps with work. It does not own work. It drafts, summarizes, suggests, and answers, and then a human decides what happens next. This is not a limitation Microsoft forgot to fix. It's the design. The moment you expect Copilot to be accountable for an outcome, the way an employee or a piece of automation software is accountable, you've left its territory.

"For a person." Copilot activates when a human asks it something. It is not watching your inbox at 6 a.m., noticing that a shipment file arrived, and processing it. Somebody has to be present, in the app, making a request. (Microsoft's agent technologies loosen this boundary in specific, deliberate ways, and Chapter 2 places them properly. For the Copilot in your Word and Outlook, the boundary holds.)

"Working inside." Copilot assists the work happening in the app in front of you. It has real reach into your organization's Microsoft content, but it is not an operations center monitoring your queue, your vendor portal, and your tracking spreadsheet simultaneously. It sees

what its position lets it see.

"A Microsoft document or conversation." Copilot's native ground is the Microsoft 365 world: your email, your Teams messages, your documents, your SharePoint content. The moment your work depends on the ERP system, the industry database, or the application only your longest-tenured admin knows how to use, you're at the edge of the map.

Here's the sentence doing real work, on two requests from the same operations manager on the same afternoon:

"Catch me up on the vendor thread. What was decided, and is anyone waiting on me?" A person, present, asking about a Microsoft conversation. Every boundary satisfied. Copilot answers it on request.

"Whenever a PO arrives in the shared inbox, validate it against the ERP and update the tracker." No person present per instance, work living partly outside Microsoft content, and accountability for an outcome. Three boundaries crossed. This is a real, solvable problem, and the back of this guide is about solving it. It is not a Copilot prompt, and no phrasing will make it one.

Keep the sentence somewhere you can see it. When Copilot delights you, the task fit inside all four boundaries. When it disappoints you, check which boundary the task crossed. It's rarely a mystery for more than a minute.

What the assistant is good at, demonstrated

Within its territory, Copilot's wins cluster into four families: **catching you up** (threads, meetings, documents compressed into orientation), **first drafts** (the email, the update, the proposal skeleton), **answering from your own content** (what did we decide, where is the latest version), and **working material over** (rewrite, restructure, extract, critique).

Notice that each is a task you already do that a competent assistant sitting next to you could help with. That's the test, and it beats any feature list.

But the quality of the help tracks the quality of the request, so watch the same task done twice. A manager has a twelve-page vendor proposal to react to by 2 p.m.

"Summarize this document."

> **The proposal outlines a three-phase implementation of the vendor's platform, covering discovery, deployment, and optimization, with associated timelines and pricing...**

Accurate, generic, and useless for deciding anything. Now:

"Summarize this proposal for a skeptical reader. What exactly are they committing to deliver, what does it cost fully loaded, and where is the language vague?"

> **They're committing to configure the platform and build two integrations, described in Sections 2 and 3. Training is listed as "up to four sessions," which is a ceiling, not a promise. The first-year cost comes to $84,000 when you add the $24,000 services line to the $60,000 license; the proposal doesn't state year-two pricing. Vague spots: "ongoing optimization support" never defines hours, and the integration timeline "assumes timely client input" without saying what timely means.**

Same document, same AI, sixty seconds apart, and the second answer is a meeting you no longer need to schedule. It isn't perfect, either: it never flagged that the success metrics are listed as "to be mutually agreed," which my own read caught in Section 6. That's the division of labor working as designed, a strong first pass plus a human who still

reads. Chapter 5 turns this difference into five patterns you can run on anything; for now, just register that the gap between "Copilot is generic" and "Copilot is useful" is usually the request, not the product.

What Copilot is not

Now the other half, because nobody selling the license leads with it.

Copilot is not an automation engine. It will not watch for events and act on them. If your problem statement starts with "every time X happens," you are describing automation, a different tool family with its own place in this guide.

Copilot is not a system of record. It doesn't maintain your data, enforce your business rules, or guarantee Friday's answer matches Monday's. Anything where consistency and auditability are the requirement (approvals, calculations feeding decisions, compliance-sensitive records) belongs in an application, not an assistant.

Copilot is not a mind reader about your business. It knows your content, not your intent or your standards. The proposal example above is the proof in both directions.

Copilot is not a strategy. This one costs organizations the most. "We have Copilot" has a way of becoming the answer to every AI question in the building, which quietly files real problems, like that PO workflow, under "solved." They are not solved. They are unaddressed, with a license fee attached.

It will be wrong, confidently

One behavior deserves its own section rather than a subordinate clause, because it's the thing most likely to burn you in month one.

Copilot delivers correct answers and incorrect answers in exactly the same tone. It does not hedge more when it's less sure. Ask what the

travel policy says, and it may summarize the 2022 version that nobody deleted, fluently and helpfully, with no signal that anything is off. Ask what was decided in Tuesday's meeting, and it may state a number that's approximately right, which for numbers means wrong.

The countermeasure is a norm, not a feature.

> **WARNING: Anything that feeds a decision gets verified at the source.**

In most grounded answers, Copilot shows the documents and messages it drew from, though not uniformly in every app. Where the sources appear, clicking through is the entire discipline, and it costs seconds; where they don't, find the source the old way before you act. Where the stakes are money, commitments, or someone's employment, verification is not optional courtesy; it's the process. Part III explains *why* wrong answers happen and how to make them rarer by fixing what Copilot draws from. But no cleanup eliminates the need for the check. The human who acts owns the action, and that principle runs from here to the last page.

The expectation conversation

A practical move that costs nothing: before your team gets access, or this week if they already have it, have the five-minute expectation conversation. It sounds like this:

Copilot is an assistant, not an oracle. It will save you real time on catching up, first drafts, and finding things in our own content. Its first answer is a draft, and you own everything you send. It states wrong answers as confidently as right ones, so anything feeding a decision gets checked at the source. And when you find a task it's surprisingly good at, tell the rest of us.

Teams that hear that paragraph calibrate in a week. Teams that don't spend a month oscillating between "this is magic" and "this is useless," and both moods produce bad decisions. The magic mood ships unreviewed output with the company's name on it. The useless mood abandons a valuable tool because it was graded against a promise nobody who understood it ever made. The useless-mood month in the opening pattern is exactly what one reset conversation prevents.

From here, the guide follows the order of need: what the Copilot family actually consists of and which one you have, then the daily craft, then the content underneath it all, then rollout, and finally the edges of the map, where that PO workflow gets its real answer.

One chapter down. The most important sentence is already behind you.

CHAPTER 2

The Copilot Family, Untangled

IN THIS CHAPTER

- The one question that sorts every Copilot-branded product
- The three categories: general assistant, work-grounded assistant, and agents
- A two-question test that survives Microsoft's renames
- The invoice, explained: the shape of the bill, and the principles that outlive any price list

"Copilot" is not a product. It's a brand name that Microsoft drapes over a growing family of AI features, and the family members differ in ways that matter enormously: what they can see, what they cost, and what they're for.

This causes real confusion with real consequences. It goes like this: a team concludes that "Copilot can't see our documents" when they're using the included chat that isn't supposed to. An executive questions the invoice because "everyone already has Copilot in Windows, why are we paying?" Or the opposite: a team pays for the full product and uses it exactly like the included one, which is like buying a truck and only ever using the radio. And if your license simply appeared one Tuesday, this chapter is also where the invoice finally gets explained: which tier you're holding, and what the money actually buys.

Microsoft renames and reshuffles this lineup regularly, so this chapter is deliberately built around the categories rather than the current logos. Product names will drift. The categories have held steady, and once you can place any Copilot-branded thing into its category, you'll know what it can do before you've read a word of the announcement.

The question that sorts everything

When you encounter anything called Copilot, ask one question first:

What content is it grounded in?

"Grounded" is the term for where an AI assistant gets its information beyond its general training. It is the single most important word in this chapter, and honestly one of the two or three most important in the guide. Grounding determines what the assistant knows about *you*, and therefore what it can actually do for you.

There are three answers, and they define the family tree:

1. **Grounded in the web and general knowledge.** The assistant can reason, write, and search the internet, but it knows nothing about

your organization.

2. **Grounded in your work content.** The assistant can additionally see your email, files, meetings, and chats, subject to your permissions. This is where the workplace value lives, and it's what you're paying for when you pay.
3. **Grounded in something specific on purpose.** The assistant was pointed at a defined set of content or systems to do a defined job. This is the agent category, and it's built rather than bought.

Every Copilot you'll ever meet is one of these three. Let's meet them.

Category one: the general assistant

This is the included tier of the family: the consumer Copilot on the web and in Windows, and the business chat that rides along with many Microsoft 365 subscriptions at no additional license cost. It's a capable general-purpose AI assistant. It writes, summarizes what you paste into it, brainstorms, and searches the web.

What it cannot do is see your work. Ask it "summarize my meetings this week" and it has no idea you have meetings. Ask it to draft an email in your style and it has never read a word you've written. It's a smart stranger.

That does not make it useless. For general drafting, thinking through a problem, or working with content you paste in, the included tier is genuinely productive, and for some roles it's plenty. The important thing is calibration: if this is what your team has, their "Copilot experience" tells you almost nothing about what the paid product would do. Judging Microsoft 365 Copilot by the included chat is judging a colleague by a stranger with the same name.

One caution belongs here. The business chat that comes with your organization's subscription typically includes commercial data protection, meaning your prompts aren't used to train models and stay within your organization's boundary. The consumer versions on

personal accounts may not, and free software tends to charge somewhere; with consumer AI chat, the currency can be your data. Chapter 13 picks this thread up properly.

> **WARNING: Know which Copilot you're typing into before you paste anything confidential.**

Category two: the work-grounded assistant

This is Microsoft 365 Copilot, the flagship, the one with the per-user monthly price that made your finance team ask questions. It's the same style of assistant as category one with one transformative difference: it's grounded in your organization's Microsoft 365 content.

That means it sits inside Word, Excel, PowerPoint, Outlook, and Teams, and it can draw on your emails, files, chats, and meetings, filtered through your permissions, to do its work. "Catch me up on the vendor renewal project." "Draft a reply that reflects what we agreed in yesterday's meeting." "What did the team decide about pricing?" Category one cannot touch any of those. Category two does all three on request.

Watch the difference on one question, asked of both categories on the same morning:

"Summarize my meetings this week and flag anything I owe people."

The general assistant:

> **I don't have access to your calendar or meetings. If you paste in your meeting notes or a list of your meetings, I'm happy to summarize them and pull out action items.**

The work-grounded assistant:

> **You had six meetings this week. Two left items with you: Tuesday's vendor review ended with you owing the procurement lead the revised budget by Friday, and Wednesday's ops sync assigned you the first draft of the Q3 staffing note. Thursday's client call set a follow-up for the 24th with no deliverable on your side. The other three closed with nothing assigned to you.**

Same question, same morning, and only one answer is useful. That is what the license buys. Not a smarter AI: an AI that can see your work.

Everything in Part II of this guide, the daily wins, assumes this category. And everything in Part III exists because of one fact about it: **the work-grounded assistant is exactly as good as the content it's grounded in.** Feed it a well-organized environment and it looks brilliant. Feed it ten years of duplicate files, stale documents, and chaotic permissions and it will confidently summarize the chaos. Same license, wildly different value. Hold that thought; it gets two full chapters.

The permission point deserves one more sentence, because it reassures and warns in equal measure: Copilot can only see what the signed-in person can see. It doesn't unlock anything new. But if your organization has been sloppy about who can see what, Copilot makes that sloppiness very easy to discover. That's Chapter 7's problem.

Category three: agents, the specialists

The first two categories are assistants waiting for a person to ask something. The third category is different in kind: **agents** are AI workers configured for a specific job, grounded in specific content, and in some cases able to run without a person present in the conversation.

An agent might answer HR policy questions from an approved document library and nothing else. It might sit in a Teams channel and handle IT triage. It might watch for a form submission and process it. Some you can build by pointing and clicking in Copilot Studio; some are prebuilt by Microsoft or vendors; some are serious development projects.

Agents are where Microsoft's story starts to overlap with the automation and custom territory we map in Chapters 11 and 12, and that's exactly where they belong in this guide: later, after the foundation. I mention them now for one reason. When someone in your organization says "I heard Copilot can automatically do X," they are almost always talking about an agent, not the assistant, and the difference matters. The assistant came with the license. An agent is something someone must decide to build, ground, test, and maintain. It's a small project, not a feature toggle.

The sorting test in practice

Here's the whole chapter as a two-question test you can run on anything Copilot-branded, today or three renames from now:

1. Can it see our work content? No: category one, general assistant, useful stranger. Yes: continue.

2. Does it help a person who's present and asking, or does it do a defined job? Person asking: category two, the work-grounded assistant, the subject of most of this guide. Defined job: category three, an agent, which someone built and someone maintains.

That's it. Pricing tiers, product names, and interface locations will keep shifting underneath you, and none of it will break this test.

The test also handles an aside worth thirty seconds: the specialist, role-specific Copilots for particular products, the sales, service, security, and finance flavors living inside Dynamics and other

Microsoft platforms. Run the two questions and they sort cleanly as category two: work-grounded assistants whose grounding is that product's data. Same question, same system.

The licensing sanity check

First, the invoice itself, because Chapter 1 promised you'd get it. Exact prices change often enough that most numbers printed in a static document are wrong by the time you read them, so what follows is the shape of the bill, which has held steady for years, with the one number that has held steady alongside it.

The work-grounded assistant is sold as a per-user monthly add-on, not a standalone product: a qualifying Microsoft 365 base plan sits underneath, so your true all-in cost is always base plus Copilot. The enterprise add-on has sat at $30 per user per month on an annual commitment since launch. A small-business tier, capped at 300 users (a ceiling on that tier, not a minimum on anything else), runs meaningfully cheaper and is frequently discounted below its own list price. Microsoft has also begun folding Copilot into bundled small-business plans at a modest premium over the base plan alone, which is worth checking before you price the add-on separately. And agent capabilities bill on consumption rather than per seat, so they're their own line item. For the numbers themselves, check Microsoft's pricing page the week you buy; the structure above is what will still be true when you do.

The principles below are what outlive any price list, and they're what actually protect your budget.

Pay for grounding, not for chat. The included tier already does general chat well. The paid license is worth it precisely for the people whose work lives in Microsoft 365 content: heavy meeting loads, heavy email, document production, anyone who regularly needs to know "what did we say about X." For someone who barely touches

that world, the license mostly duplicates the included tier at a monthly price.

License the workflow, not the org chart. The strongest rollouts start with the teams whose daily work matches the win families from Chapter 1, prove the value with real numbers, and expand from evidence. "Everyone gets it on day one" produces exactly the stalled-pilot pattern that Part IV of this guide is designed to prevent.

Count the agents separately. Agent capabilities often carry their own licensing and consumption costs distinct from the per-user assistant license. If your plans include category three, price it as its own line item, because it is one.

If you take a single sentence from this chapter into the rest of the guide, take the grounding question. What content can this thing see? Everything else about Copilot's family tree is decoration on that answer.

Next: what the work-grounded assistant actually does all day. We start where most people live, in Outlook and Teams.

CHAPTER 3

Copilot in Outlook and Teams

IN THIS CHAPTER

- The Outlook moves: catch-up, drafting, tone coaching, and grounded search
- The Teams moves: recap interrogation and channel catch-up
- The four honest limits the demo skips
- The two-week calibration that builds real judgment

If Microsoft 365 Copilot is going to pay for itself anywhere, it's here. Outlook and Teams are where communication piles up, and communication pileup is the problem Copilot solves most reliably. This chapter covers the moves worth building into your day, the prompts that make them work, and the honest limits nobody puts in the demo.

A note on how I'll describe things: buttons move. Microsoft relocates and renames interface elements constantly, so I'll describe capabilities and show current screenshots rather than promising that a button lives in a particular corner. If something has moved by the time you read this, the capability is still there. Check Microsoft's support pages, or ask Copilot where it went and verify against what's actually on your screen; a moved button is a fair question, and also, as this chapter will show, exactly the kind Copilot answers confidently whether it's right or not.

Outlook: the inbox stops winning

Thread catch-up. The foundational move. A long thread arrives, or you return from two days out, and instead of archaeology you ask Copilot to summarize. What you get is a compressed account of who said what, where the disagreement is, and what's being asked of whom.

The skill is in what you ask for. "Summarize this thread" works, but you'll get more from:

"Summarize this thread. What decisions were made, what's still open, and is anyone waiting on me?"

That last clause is the one that matters. A summary is nice; knowing your obligations is actionable. Make "is anyone waiting on me" a standard part of your catch-up prompt and Copilot becomes a commitment detector, which is worth more than a summarizer.

Drafting and replying. Copilot will draft a reply from a short instruction: *"Draft a reply declining the Thursday meeting, propose next Tuesday, keep it warm."* The draft arrives in seconds and it's usually 80% right, which is exactly what a first draft should be. Adjust tone, add the sentence only you would know to add, send.

Two habits separate people who get value here from people who embarrass themselves. First, never send without reading; Chapter 1's assistant model applies with no exceptions. Second, give the draft your facts. Copilot knows the thread; it doesn't know that the client called you yesterday. The instruction *"Draft a reply. Context: the client already approved the budget by phone yesterday"* produces a dramatically better draft than hoping the AI intuits what it cannot know.

> **WARNING: Never send Copilot's draft without reading it; you own every word that leaves under your name.**

Tone coaching. Before sending something delicate, ask Copilot to review it: *"Read this draft. How might a frustrated customer receive it? What would you soften?"* This is a genuinely underused move. It costs fifteen seconds and it has a way of saving working relationships, because it gives you the thing email has always lacked: a second reader before the point of no return.

Finding things. "What did procurement say about the renewal terms?" "Find the email where we agreed on the delivery date." Grounded search across your own mail is quietly one of the license's best returns, especially for anyone whose job includes the phrase "let me find that and get back to you."

Teams: meetings develop a memory

The recap. If your organization records and transcribes meetings, Copilot's meeting recap is the single most impressive thing most users

will see in their first month: a summary of what was discussed, decisions reached, and action items with names attached, available shortly after the meeting ends. Newer versions also draw on the meeting chat alongside the transcript, which catches the links and side comments posted while someone else was talking.

The prompt-level move is interrogating a meeting like a document: *"What did we decide about the launch date?" "What concerns did finance raise?" "List every action item assigned to me with deadlines."* You can do this for meetings you attended, and, more powerfully, for ones you skipped.

Which leads to the cultural shift worth naming: once recaps are reliable, *attending* stops being the only way to stay informed, and some meetings stop needing to exist at all. A status meeting whose entire output is captured in a recap is a meeting auditioning for retirement. Hold that thought for Chapter 9, where we'll retire it properly.

Catching up mid-meeting. Joining late, Copilot can tell you what's been covered so far, whether your name has come up, and whether anything's been assigned to you, all without interrupting anyone. Small feature, disproportionate dignity savings.

Chat and channel catch-up. The Teams equivalent of thread summarization: *"Summarize this channel since Monday. What needs my attention?"* For anyone returning from vacation, this move alone changes the re-entry experience from a half day of scroll to twenty minutes of reading.

Drafting posts. Same rules as Outlook drafting: give it the intent and your facts, review before posting, add the sentence that sounds like you.

The honest limits

Now the part the demo skips, because you'll hit all of these in week one and I'd rather you hit them calibrated.

Recaps need transcription. No transcript, no recap. If your organization hasn't enabled recording and transcription, or people decline it, the flashiest Teams features simply don't happen. There are legitimate privacy and policy reasons a company might restrict this; Chapter 13 covers how to think about them. Just know that "Copilot doesn't work in our meetings" is usually a settings-and-policy statement, not a product one.

Summaries flatten. A recap tells you what was said. It does not reliably capture that the room went quiet when the date was mentioned, or that the client's "fine" was the dangerous kind of fine. Copilot compresses content, not subtext. For meetings where the subtext was the meeting, treat the recap as the minutes, not the truth.

Confidence is not accuracy. Copilot answers questions about your mail and meetings in the same assured tone whether it's right or approximately right. For anything consequential (a commitment, a number, a quote you'll attribute to someone), click through to the source before you act. The source links it provides in most answers exist precisely for this; use them, and when they're absent, check the old way.

Attribution can smear. In fast multi-speaker meetings, action items occasionally land on the wrong person, and transcription mangles names, especially non-English ones. The recap is a draft of reality. A human confirms who owns what, which is a two-minute habit that Chapter 9 builds into the workflow.

Building the habit

Here's the two-week calibration I recommend to every team, and it's deliberately unambitious. Pick two moves from this chapter, not five. For most people I suggest thread catch-up with the "waiting on me"

clause, and meeting recap interrogation. Use them on real work every day for two weeks.

Something predictable happens in that window: you learn where Copilot is trustworthy for *your* work, in *your* organization, with *your* content. That calibration is personal and it cannot be transferred by training slides. After two weeks, add the next move. People who try everything in the first three days form vague impressions; people who drill two moves form judgment.

And notice what you're actually building. Not familiarity with a feature, but a changed default: pileup that used to cost you an hour now costs minutes, and the hour goes somewhere better. That reallocation, multiplied across a team, is the entire business case for the license. We'll count it properly in Chapter 10.

Next, the content side of the house: what Copilot does inside Word, Excel, and PowerPoint, including the app where it still disappoints and what to do about it.

CHAPTER 4

Copilot in Word, Excel, and PowerPoint

IN THIS CHAPTER

- Why Word is Copilot's strongest surface, and the referencing move most users miss
- The document-first workflow that makes PowerPoint generation actually work
- The honest truth about Excel, and the two workarounds
- The three-condition pattern that predicts which features will deliver

Communication was Chapter 3. This chapter is production: documents, spreadsheets, and decks, the artifacts your organization actually ships. Copilot's help here ranges from genuinely excellent to honestly frustrating depending on the app, and I'll tell you which is which, because calibrated expectations are the difference between a tool you use and a tool you tried once.

Word: the blank page is over

Word is Copilot's strongest production surface, and the reason is structural: writing is what large language models do natively. Everything else is adaptation; this is home.

Drafting from intent. Give Copilot a brief and it returns a structured draft: *"Draft a two-page project proposal for replacing our inventory intake process. Audience: the operations VP. Include current pain, proposed approach, timeline, and risks."* What arrives will be organized, competent, and generic, which is precisely what a first draft is for. Your job moves from producing words to correcting them, and correcting is faster. The people disappointed by Word Copilot are usually judging the draft as a final product. It isn't one, and treating it as clay rather than marble is the whole skill.

Drafting from your own files. The stronger version of the same move: point Copilot at existing documents as raw material. *"Draft a statement of work based on [the discovery notes] and [last year's SOW*

for the Harmon project]." Now the draft arrives pre-loaded with your terminology, your structure, and your facts instead of the internet's average. This referencing move is one of the license's best returns in Word, and most users never discover it.

Working a document over. Once text exists, Copilot manipulates it well: tighten this section, convert these paragraphs to a table, adjust the tone for an external audience, add a summary up top. And the interrogation moves from Chapter 3 work on documents too. Before an important document leaves your hands: *"What questions would a skeptical reader ask after reading this? What am I asserting without support?"* That prompt improves deliverables more reliably than any editing technique I know.

Summarizing what you receive. The forty-page vendor contract, the consultant report, the policy update. Ask for the summary, then ask the questions that matter to you: what are the termination terms, what changed from the last version, what's unusual here. Reading for orientation is now optional; reading for verification never is, and for contracts, the verification reading is the one that counts.

PowerPoint: a strong start, not a finished deck

PowerPoint Copilot's headline act is generating a deck from a prompt or, better, from a Word document. Write the narrative in Word first, then have Copilot build the deck from it. The document-first workflow produces dramatically better results than conjuring slides from a one-line prompt, because the thinking exists before the slides do, which incidentally is how good decks have always been made.

What you'll get is a real starting structure: sections, headlines, layouts, speaker notes. What you won't get is your brand, your emphasis, or design judgment. Expect to rework the visuals and cut a third of the slides. And one honest subtraction: if your organization runs a heavily customized brand template, Copilot's generated decks tend to fight it,

breaking layouts in ways that can cost back the time they saved. In branded environments, use it for the narrative (structure, headlines, speaker notes) and build the visuals inside your own template. The honest accounting is that Copilot removes the first two hours of deck-building, the structural hours, and leaves the final hour of judgment to you. That's a good trade; just don't schedule as if it removed all three.

Two smaller moves earn their keep: summarizing a deck you've received ("what's the argument of this presentation, and where's it weakest?") and generating speaker notes for slides that lack them.

Excel: the honest chapter section

I'm going to be more candid about Excel than the marketing is, because Excel is where the gap between demo and Tuesday is widest, and where the most disillusioned users come from.

Here's the core issue: Excel Copilot works well on *clean, structured tables* and struggles with the spreadsheets your organization actually has. Real-world workbooks are archaeology: merged cells, headers three rows down, notes typed into data columns, fourteen tabs of history, formatting doing the job of structure. Copilot needs to know what the data *is* before it can reason about it, and archaeology defeats it. Users point Copilot at a legacy workbook, get a confused or refused response, and conclude the feature is useless. The feature isn't useless. It's fussy about inputs in a way the demos never show, because demo data is always clean.

So the practical guidance comes in two halves.

Make your data legible, then Copilot performs. First, the prerequisite that explains most "it's grayed out" complaints: the workbook has to live in OneDrive or SharePoint with AutoSave on, because Copilot largely ignores local files. Then convert ranges to proper Excel tables, one header row, one data type per column, no

merged cells in the data area. On legible tables, Copilot genuinely delivers: highlight what's interesting here, add a column calculating margin, show me total by region, why did Q3 dip. For analysis-shaped questions on well-shaped data, it's a capable junior analyst.

Use it as a formula consultant everywhere else. Even on messy workbooks, Copilot is excellent at the meta-work: *"Write a formula that pulls the latest date per customer." "Explain what this formula does in plain English." "Why is this returning an error?"* Formula help alone justifies the feature for most business users, because formulas are precisely the part of Excel most professionals never fully learned and always felt they should have.

One warning with teeth: **verify numbers that feed decisions.** Copilot in Excel can mischaracterize data or build a subtly wrong calculation while narrating confidently. For anything that reaches a budget, a price, or a report upward, check the math the way you'd check a new hire's first model. Chapter 1's rule (it's an assistant, you own the output) applies to numbers hardest of all, because wrong numbers compound in ways wrong sentences don't.

> **WARNING: Verify any Copilot number that feeds a budget, a price, or a report before you rely on it.**

The pattern across all three apps

Look back across this chapter and Chapter 3 and a shape emerges. Copilot performs best when three things are true: the raw material is text-like, the raw material is *yours* (grounding beats generality every time), and a human's judgment finishes the work. Word hits all three constantly, which is why it shines. PowerPoint hits two. Excel hits them only when the data is structured, which is why your mileage varies tab by tab.

As a scorecard:

	Text-like material	Grounded in your content	Human finishes the work	Net
Word	Yes	Yes	Yes	Shines
PowerPoint	Yes	Partly	Yes	Strong start, you close
Excel	Only as structured tables	Yes	Yes	Varies tab by tab

That shape is worth internalizing because it predicts the future better than any feature list: as Microsoft ships new capabilities, the ones that follow this pattern will work, and the ones that promise to skip the human's finishing judgment will disappoint. You now have a way to read the announcements.

Next chapter: the prompting patterns themselves. You've seen a dozen good prompts scattered through the last two chapters. It turns out they're all variations of five moves, and five is a learnable number.

CHAPTER 5

Prompting for Normal People

IN THIS CHAPTER

- The four ingredients of every request that works
- The five patterns: Catch Me Up, Draft From Intent, Transform, Interrogate, Coach Me
- Before-and-after examples for each
- How to make the patterns stick with a team

There's an entire cottage industry devoted to making this topic complicated. Prompt engineering courses, hundred-item prompt libraries, threads promising the ten secret phrases that unlock AI. You need none of it.

Here's what you actually need to know: Copilot's output quality tracks the quality of your request almost one to one, and good requests follow patterns you can count on one hand. Office work runs on five of them. You've already seen all five in action across the last two chapters; this chapter names them so they stick.

The anatomy of a request that works

Before the patterns, the parts. Every effective prompt contains some mix of four ingredients:

Goal. What you want produced. Be concrete: "a one-page summary," "a reply that declines politely," "a table with three columns."

Context. What Copilot can't know: the audience, the situation, the facts that live in your head or your phone calls. This is the ingredient people omit most, and its absence explains most generic output. Copilot knows your content; it does not know your intent.

Constraints. Length, tone, what to include, what to leave out. "Under 200 words." "No jargon." "Don't mention pricing yet."

Source. What material to work from: this thread, that document, the meeting from Tuesday. Grounding the request in specific content is

what separates the paid Copilot from a generic chatbot, so use it.

You won't need all four every time. But when output disappoints, the fix is almost always one of these four ingredients missing, and now you have a checklist instead of a shrug.

Pattern one: Catch Me Up

The move: compress a pile of communication into orientation plus obligations.

The skeleton: "Summarize [source]. What was decided, what's open, and what needs something from me?"

The obligation clause is what elevates this from a book report to a working tool, and it's the single highest-frequency prompt in office life. Variants: "Summarize this channel since Friday." "What did I miss in this meeting, and was I assigned anything?" "Across my email from this week, what commitments have I made?"

Before: "Summarize this thread." → A paragraph of who-said-what you still have to think about.

After: "Summarize this thread. What was decided, what's still contested, and is anyone waiting on me?" → A decision list, an open-issues list, and your to-dos. Same feature, different day.

Pattern two: Draft From Intent

The move: turn a goal plus your private facts into a first draft.

The skeleton: "Draft [thing] for [audience]. Goal: [outcome]. Context: [the facts only you know]. Tone: [tone]. Keep it [length]."

The context line is the difference between a draft you can send after light edits and a draft that reads like a template. Copilot was not on yesterday's phone call. Tell it.

Before: "Write an email to the team about the deadline." → Serviceable, hollow, could be from anyone at any company.

After: "Draft an email to the project team. Goal: move the deadline from the 15th to the 22nd without triggering panic. Context: the client requested the change, this is good news for QA, and the QA lead already knows. Tone: calm, brief." → A draft that's 90% sendable.

Pattern three: Transform

The move: content exists in one shape; you need it in another.

The skeleton: "Turn [this] into [that], keeping [what matters]."

Notes into a status table. A rambling document into an executive summary. Bullet points into prose, prose into bullets, a Word narrative into a deck, meeting chaos into an agenda for next time. Transformation is mechanically easy for AI and mechanically tedious for humans, which makes it the pattern with the best effort-to-payoff ratio in the whole book.

Before: "Clean this up." → Copilot guesses what clean means.

After: "Turn these raw notes into a status table with columns for workstream, owner, status, and blocker. Anything unclear goes in a questions list below the table." → Exactly the artifact you needed, plus a list of what the notes failed to capture.

Pattern four: Interrogate

The move: ask questions *of* content instead of reading all of it, and ask questions your own attention would miss.

The skeleton: "Based on [source]: [your question]." And the advanced form: "What's missing from this? What would a [skeptical reader / lawyer / frustrated customer / CFO] challenge?"

Interrogation is the pattern that turns Copilot from a producer into a second pair of eyes. What are the termination terms in this contract? Which action items from last month's meetings are still unresolved? What am I asserting in this proposal without evidence? The persona variant ("read this as a skeptical CFO") is the closest thing to a secret weapon this guide will admit to, because it manufactures the reviewer you don't have.

Before: reading a 40-page vendor agreement start to finish, retaining maybe a tenth.

After: "From this contract: summarize termination rights, list every deadline that binds us, and flag anything unusual compared to a standard services agreement." Then you read those clauses yourself, because verification is still your job. Orientation delegated; judgment retained.

Pattern five: Coach Me

The move: get feedback on your work before the audience does.

The skeleton: "Review this [thing]. How will [audience] receive it? What would you strengthen, soften, or cut?"

This is the least used pattern and the one I'd defend hardest. It costs fifteen seconds, applies to everything (emails, proposals, decks, difficult Teams messages), and it fills a permanent gap in professional life: the absence of a candid reader before the moment of no return. It also compounds; ask "what would you cut" enough times and you start pre-cutting, which means the AI made you a better writer by critique rather than by replacement. That's the direction you want this relationship to run.

Before: send, then wonder.

After: "Review this reply to a client who's upset about the delay. Where could my tone read as defensive? What's the one sentence to

add that would rebuild trust?" Then send.

One rule sits underneath all five patterns, and it's Chapter 1's rule wearing work clothes:

> **WARNING: The patterns change how fast the work gets done, not who answers for it.**

Making five patterns stick with a team

Two closing notes for anyone rolling this out beyond themselves, previewing Part IV.

First, teach patterns, not prompts. A prompt library goes stale and nobody browses it under deadline. Five named patterns fit on one page, cover most office work, and generate infinite specific prompts on demand. (That one page is Appendix A of this guide, and it's deliberately built to be forwarded.)

Second, follow-up is not failure. The first response is a first response; "shorter," "more formal," "now as a table," "you missed the budget angle" are how the tool is meant to be used. New users often quit after one mediocre answer, graders where they should be editors. The habit to model out loud is conversational: nudge, refine, accept. People who treat Copilot as a vending machine get vending machine results. People who treat it as an assistant get an assistant.

That closes Part II. You now know what the assistant does all day and how to ask well. Part III turns to the uncomfortable question underneath all of it: why the same license produces brilliance in one company and nonsense in another. The answer is your content, and it's fixable.

CHAPTER 6

Grounding: Where Copilot's Answers Come From

IN THIS CHAPTER

- How grounded answers actually get made: retrieval first, writing second
- The four ways grounding goes wrong
- Why content quality is the highest-leverage Copilot investment
- A five-minute self-diagnosis for your own tenant

Two companies buy identical Copilot licenses. Six months later, one describes it as transformative and the other calls it a disappointment they're about to cancel. Same product, same price, same Microsoft. This divergence is common enough, and consistent enough, to say plainly: the difference is almost never the AI, the training, or the enthusiasm of the team.

It's the content. This chapter explains why, in plain language, because once you understand the mechanism you'll know exactly what to fix, and Chapter 7 will fix it.

How Copilot actually answers you

When you ask the work-grounded Copilot a question, something happens before any AI gets involved: a search. Copilot goes looking through the content you have permission to see, your emails, files, chats, and meetings, and retrieves the pieces that seem most relevant to your question. Then, and only then, the AI reads those retrieved pieces and composes an answer from them.

That two-step is worth restating because everything in Part III follows from it: **Copilot doesn't know your organization. It looks things up, fast, and then writes about what it found.**

The AI's writing ability is essentially constant; Microsoft supplies that, and it's the same for everyone. The lookup, though, runs against *your* content, and your content is not constant. It's the variable. Which means the quality equation for grounded answers actually looks like one:

```
answer quality  =  constant writer  x  your variable content
```

Or in plain words: a brilliant writer, working from whatever your organization left lying around.

If what's lying around is current, well-organized, and correctly permissioned, the writer looks like a genius. If it's three conflicting versions of the pricing sheet, a policy from 2019 nobody deleted, and a folder named "FINAL_v7_use_this_one," the writer will confidently synthesize garbage. The demo you saw ran on a curated environment. Your Tuesday runs on your real one.

The four ways grounding goes wrong

Stale content wins searches. Retrieval finds relevant content, not correct content. The 2022 travel policy is extremely relevant to a travel question; it's also wrong, and Copilot has no way to know your organization stopped following it.

> **WARNING: Old content doesn't sit there harmlessly anymore; it competes with the truth, and often wins.**

Duplicates split the signal. Five copies of the proposal in five places, each slightly different, means retrieval may grab any of them, and answers wobble depending on which copy won. People experience this

as "Copilot is inconsistent." Copilot is perfectly consistent; your content isn't.

Structure is invisible ink. A wall of text titled "Notes" retrieves worse than a clearly titled document with headings that say what each section is. Retrieval and the AI both lean on the signals humans lean on: names, titles, headings, dates. Content that's hard for a colleague to skim is hard for Copilot to use, which gives you a convenient rule of thumb: Copilot readability and human readability are the same property.

Permissions decide the universe. Copilot searches only what the asking person can access. This is the security model working as designed, and it cuts both ways. Locked-down content that should be shared produces answers with holes ("Copilot doesn't know about our project" usually means "our project lives in a private site"). And overshared content that should be locked down gets found, which is such an important failure that it opens Chapter 7.

Why this is actually good news

It would be worse if the divergence between companies came from the AI, because you can't fix Microsoft's model. You can absolutely fix your content, and here's the part I lean on with every client who's deflated by their early Copilot experience: **your organization's content quality is now visible, measurable, and improvable, and improving it upgrades every Copilot answer for every user simultaneously.**

Think about what that means for leverage. Training helps one user at a time. Prompting patterns help one request at a time. Content cleanup helps every request from every user from now on. It is, by a wide margin, the highest-leverage Copilot investment available to most organizations, and it costs process and attention rather than licenses.

There's a bonus that predates AI entirely: every fix in Chapter 7 (current content, fewer duplicates, honest permissions, findable structure) was worth doing in 2015. Organizations just couldn't feel the pain concretely enough to act. Copilot converts diffuse "our SharePoint is a mess" guilt into specific, visible wrong answers, and specific visible pain gets budget. Use that.

A quick self-diagnosis

Before the cleanup chapter, run this five-minute test in your own tenant. Ask Copilot five questions your team genuinely asks each other: what's our current pricing for X, what's the process for onboarding a client, what did we decide about Y, where's the latest version of Z, what's our policy on W.

Score each answer: right, wrong-but-plausible, or "it couldn't find it." Then look at *why*, using the four failure modes above, because the pattern of failures is your cleanup priority list. Mostly stale answers means an archiving problem. Wobbly answers mean duplication. Holes mean permission walls. Wrong-but-plausible everywhere means all of the above.

Bring that scorecard to the next chapter. It's about to become a project plan, and a mercifully finite one: the goal is not a perfect tenant, it's a tenant where the content Copilot retrieves most often deserves to be retrieved.

CHAPTER 7

The Content Cleanup That Pays for Itself

IN THIS CHAPTER

- Why the oversharing audit comes first, and how to run it
- Retiring stale content by impact, not completionism
- Making good content findable
- The three light habits that keep it clean

This is the chapter your Copilot investment has been waiting for, and I'll be blunt about why it exists: most organizations bought the license, skipped this work, and are now grading the AI on a decade of deferred housekeeping. The good news is that the cleanup is finite, sequenceable, and delivers value even in the first week. Here's the order of operations I use with clients, arranged so the riskiest problem gets handled first.

```
The cleanup, in order

  1. Close the oversharing holes    risk first, this week
  2. Retire the stale content       by impact, not completionism
  3. Make good content findable     names, structure, one home
  4. Make it stay clean             owners, cadence, defaults
```

The order is the claim: one and two remove what can hurt you, three improves what remains, four keeps you from being back here in eighteen months. Now the steps in detail.

Step one: close the oversharing holes (do this first)

Before Copilot, badly permissioned content was protected by obscurity. The salary spreadsheet shared with "Everyone" in 2021 was technically exposed, but nobody stumbled into it because nobody searches for what they don't know exists.

That protection is over. Copilot searches everything the asking person can access, which means what was always technically visible now

actually gets found. Ask an innocent question and an overshared document can arrive in the answer, neatly cited. Nearly every organization has some version of this exposure, and it's the one Copilot problem that can genuinely hurt you, so it goes first, before the tidying.

> **WARNING: Copilot ends security by obscurity: it will politely surface anything people could technically always see.**

The practical sequence:

1. **Hunt the broad grants.** Content shared with "Everyone," "Everyone except external users," or org-wide groups is your priority list. Microsoft's admin tooling (SharePoint admin reports, Purview, and the oversharing assessment tools Microsoft has shipped specifically because of Copilot) will surface these. Your IT admin will know the current tool names; the ask from you is simply "show me everything effectively visible org-wide."
2. **Triage by sensitivity, not volume.** You're not fixing all of it this month. You're finding the HR, finance, legal, and personal-data content in that list and fixing *that* this week.
3. **Fix the source, not the symptom.** Restrict the site or library, don't play whack-a-mole with files. Most oversharing traces to a handful of sites created permissive years ago.
4. **Change the default going forward.** New sites and shares start restrictive. Oversharing is a faucet, not a puddle; turn the faucet.

One stopgap worth knowing while the cleanup runs: Microsoft ships a restricted-search mode for SharePoint that temporarily limits what organization-wide search and Copilot can draw from to a short, approved list of sites. It's a tourniquet, not a fix, and it blunts Copilot's usefulness while it's on, but it converts "we can't roll out until permissions are perfect" into "we can roll out while we fix them."

Your admin will know it by its current name.

If your organization does exactly one thing from this entire guide at the admin level, this is the thing.

Step two: retire the stale content

Chapter 6 established the mechanism: old content competes with the truth, and often wins retrieval. The countermeasure is archiving, and the trick is doing it by impact rather than by completionism.

Start where wrong answers hurt: policies, pricing, processes, templates, anything a person or a Copilot might act on. For each, the question is simple: if this document came up in an answer today, would we stand behind it? No: archive it, delete it, or mark it superseded with a pointer to the current version.

Archive, don't hoard-in-place. Moving old content to an archive location (out of the main search scope) preserves it for compliance while removing it from competition. Deleting is better where retention rules allow, but archiving is the achievable ask.

Kill the parallel truths. Where five versions of a document coexist, one becomes canonical and the rest get archived. The canonical copy gets an authoritative home and, ideally, a plain statement in the document itself: what this is, when it was last confirmed, who owns it. Those sentences are for humans and for retrieval alike.

Step three: make the good content findable

With the dangerous and the dead handled, improve the signals on what remains. This is lightweight work with outsized retrieval returns.

Names that say what things are. "Q3 Pricing Guide 2026" beats "pricing_final_v7." Sites and libraries named for what they contain. This sounds trivial; it is trivial, and it's also exactly what retrieval keys on.

Structure inside documents. Real titles, real headings, a summary paragraph up top for anything important. The rule from Chapter 6 applies: skimmable for humans equals usable for Copilot, so you're improving the pre-AI experience with every fix too.

One home per topic. The pattern that serves retrieval best is boring: a known site for policies, a known library for templates, project content in project sites. Every "where does this live?" ambiguity your team feels, Copilot feels worse.

Step four: make it stay clean

Cleanup without new habits is a diet without a lifestyle; you'll be back here in eighteen months. Three light mechanisms prevent the relapse:

Ownership. Every important site or library gets a named owner whose job includes its accuracy. Unowned content is future stale content, no exceptions I've ever observed.

A review cadence. Owners confirm or archive their critical content on a schedule, quarterly for fast-moving material, annually for stable material. The calendar entry is the mechanism; goodwill is not.

Lifecycle defaults. Where your admins can set them, expiration and review policies on sites do the remembering for humans. Projects end; their sites should too.

Making the case upstairs

This work needs some IT time and some content-owner time, so here's the framing that gets it approved, drawn from the leverage math in Chapter 6: content cleanup is the only Copilot investment that improves every answer for every user at once, and its first step closes a genuine data-exposure risk that predates AI. Pair the ask with your five-question scorecard from Chapter 6 as the before picture, then rerun the same five questions after each step as the after. Watching "wrong-but-plausible" turn into "right, with the correct source cited" is

the most persuasive slide you will ever not have to design.

And notice what you've actually built by the end of this chapter: not just a better Copilot, but a more legible organization. The AI was the forcing function. The asset is yours either way.

Part III complete. Part IV turns to the humans: why rollouts stall, and the 30-day plan that keeps yours from joining them.

CHAPTER 8

Why Copilot Rollouts Stall

IN THIS CHAPTER

- The three deficiencies that stall every rollout, and the politics of fixing each
- Why optional new tools lose to mandatory old habits
- The compounding cost of a stalled attempt
- The three-question pre-flight checklist

Here's the six-month picture at a lot of organizations that bought Copilot with real enthusiasm: a fraction of licensed users touch it weekly, a smaller fraction get real value, most have quietly reverted to their old habits, and somebody in finance is circling the renewal line item. Nobody can quite say whether it worked, which in budget language means it didn't.

I've watched this movie for thirteen years with SharePoint, with Teams, and now with Copilot, and the plot never changes because the cause never changes. Rollouts don't stall on technology. They stall on three deficiencies, usually all at once, and every one of them is

preventable before day one.

Deficiency one: nobody owns it

Most Copilot rollouts have a sponsor: the executive who approved the spend. Sponsors matter, but a sponsor is not an owner. An **owner** is the person whose actual job it is to make specific work happen differently with this tool, who has authority over the affected workflow, and who is expected to report whether things improved.

Test your own rollout with one question: *who, by name, is responsible for Copilot changing how work gets done on this team?* If the answer is "IT deployed it" or "everyone, really," you don't have a rollout. You have a license distribution. IT can make Copilot available; IT cannot make the operations team run its weekly reporting differently, because IT doesn't own that workflow. Availability was never the hard part.

Naming the owner is a political act, and pretending otherwise is why it gets skipped. Three wrong owners get picked constantly. The IT lead, who can configure anything and change nothing about how the team actually works. The enthusiast, who loves the tool but holds no authority over the process. And the delegated junior, handed "AI champion" as a stretch assignment precisely because their time was cheap, which tells the team everything about how serious this is. The right owner is the person who already answers for the workflow's output: the manager whose Monday report it is, the lead whose queue it is. If that person doesn't want the job, don't work around them. Scope the pilot down until you find a workflow whose natural owner does, because an unwilling owner is deficiency one with a name attached.

Deficiency two: nothing changed

The second deficiency is subtler and deadlier: Copilot gets introduced *alongside* the existing way of working instead of *replacing* any part of it. Every task now has two paths, the familiar old one and the

unfamiliar new one, and using the new one is optional.

Optional new habits lose to mandatory old habits every single time, and deadline pressure decides the contest early. Week one, people experiment. Week three, the quarter heats up, and everyone routes down the path they can walk without thinking. The tool didn't fail; it was never given a job. It was given permission, and permission is not a workflow.

The countermeasure is retirement: a real adoption changes the process so the new way is *the* way, which means something old visibly stops. The status meeting whose content now lives in recaps stops being held. The manually assembled Monday report stops being manually assembled. The "let me find that and get back to you" stops being an acceptable answer to questions Copilot answers in seconds.

The retirement sentence has a grammar, and filling it in is a five-minute test of readiness: on a date, we stop a specific step, in a specific workflow, and a named person closes the old path. "On March 3 we stop hand-assembling the Monday report; drafts generate from the tracker, and the ops manager retires the old template." If any slot won't fill (no date, no specific step, nobody empowered to close the path), you've found the missing piece while it's still cheap, which is the entire point of writing the sentence before the money moves. If your rollout plan contains no such sentence, it isn't an adoption plan yet; Chapter 9 will make you write one.

Deficiency three: measuring novelty

Ask a stalling rollout how it's going and you'll hear activity numbers: how many people signed in, how many prompts were submitted, engagement is up. These numbers always look respectable in month one and they mean nothing, because activity is what novelty produces automatically. Every tool in history spiked on arrival.

The numbers that mean something are operational, and they require a *before*: hours the weekly report used to take, days the proposal cycle used to run, time between "question asked" and "question answered." If nobody captured the before, no after can prove anything, and the renewal conversation collapses into dueling anecdotes, which the CFO's anecdote wins.

This deficiency is the cheapest of the three to prevent. It costs one week of counting before rollout. It's also the most commonly skipped, and not only because measuring the old way feels like delay in the excitement of launch. The quieter reason is that the before-number is politically uncomfortable: counting what the report costs today can read as an audit of the people who produce it, and managers sense that instantly. The framing that defuses it is worth saying verbatim: we're measuring the workflow, not the people, and this number's only job is to give the team credit later for the improvement. Honest and rough beats instrumented and threatening. A self-reported tally on a shared sheet, kept by the people who do the work, is accurate enough to prove the case and safe enough to actually happen.

The compounding cost of stalling

One more thing before the fix, because it changes how much care the launch deserves. A stalled rollout isn't a neutral outcome you can simply rerun. It teaches the team a lesson: *AI initiatives here don't go anywhere*. That lesson lingers for years, and the second attempt starts in the hole the first one dug, needing a stronger champion and a cleaner plan just to reach the goodwill the first attempt got for free. I have watched organizations pay interest on a botched rollout half a decade later.

> **WARNING: Stalls compound: every failed attempt raises the price of the next one.**

Which reframes the stakes usefully: the question is not "how fast can we roll out Copilot" but "how do we make sure the first real test cannot end ambiguously." Fewer, better, provable. That's the design philosophy of the next chapter.

The pre-flight checklist

Everything above compresses to three questions. Answer them in writing before your rollout, or before relaunching a stalled one:

1. **Who owns this, by name,** with authority over the workflows involved?
2. **What will we stop doing** when it works, specifically?
3. **What's the before-number** we'll compare against, and who's capturing it this week?

Three honest answers and you're ahead of most of the market. Blank stares on any of the three: good news, you found the problem while it was still cheap. The next chapter turns those answers into a calendar.

CHAPTER 9

The 30-Day Team Adoption Plan

IN THIS CHAPTER

- Choosing provable workflows over valuable ones
- Week by week: baseline, run both paths, retire the old way, decide
- The retirement list, and why week three separates adoptions from trials
- The day-31 verdict: expand, fix, or kill

This chapter is a calendar. It takes one team from "we have licenses" to "we can prove whether this worked" in thirty days, and it's deliberately built for one team rather than the whole organization, because the fastest way to a successful company-wide rollout is a small, undeniable win that other teams ask to copy. Evidence recruits better than mandates.

Prerequisites from earlier chapters: an owner with authority over the team's workflows (Chapter 8), and ideally the worst of the content risks handled (Chapter 7). Have those, and the plan below runs clean.

Before day one: choose the workflows

The owner picks **two or three specific workflows**, not "email" but "the Monday status report," not "meetings" but "the weekly ops sync and its follow-ups." Good candidates share three traits: they happen at least weekly, they visibly annoy the people who do them, and they map to Copilot's win families from Chapters 3 and 4 (catching up, first drafts, answering from your content, transforming material).

Resist choosing the most valuable workflow. Choose the most *provable* one: frequent, measurable, low blast radius if the first week is rough. The point of round one is a clean verdict, and clean verdicts come from workflows where the before and after are easy to see.

Week one: baseline and setup

Capture the before-numbers. For each chosen workflow, count what it currently costs: hours per week of assembly, days of cycle time, number of "where is that?" interruptions. Precision is not required; honesty is. A simple shared sheet, filled by the people who do the work, takes under an hour and becomes the most important artifact of the month.

Have the expectation conversation from Chapter 1, verbatim if you like: assistant not oracle, first answers are drafts, you own what you send, factual claims get checked, share what works.

Teach the five patterns, not fifty prompts. One session, under an hour, built on Chapter 5: Catch Me Up, Draft From Intent, Transform, Interrogate, Coach Me, each demonstrated on the team's *actual* chosen workflows, not generic examples. Hand out the one-page pattern card (Appendix A). That's the entire training program, and it's enough, because the goal of week one is not mastery. It's a shared vocabulary and two moves per person to drill.

Week two: run both paths, watch closely

The team uses Copilot on the chosen workflows daily while the old way still exists as a safety net. This is the calibration week from Chapter 3, scaled to a team: everyone learning where the tool is trustworthy on their own work.

The owner's job this week is presence, not enforcement. A ten-minute check-in twice this week: what worked, what flopped, what prompt cracked a problem. Wins get shared immediately and specifically ("one team member's recap prompt caught an action item everyone else had missed") because peer proof converts skeptics faster than any training deck. Flops get diagnosed out loud using the four-ingredient anatomy from Chapter 5, which turns failures into technique instead of verdicts.

One more job this week: **write the retirement list.** Based on what's actually working, the owner drafts what will stop in week three. This is the "we will stop" sentence from Chapter 8, now with specifics and a date.

Week three: retire the old way

The week that separates real adoptions from tool trials. The old path closes for the chosen workflows:

- The report that Copilot now assembles is no longer built by hand; the human role becomes review and send.
- The meeting whose value was status transfer is cancelled or cut in half, with recaps and a channel post carrying the load.
- The old template, the old manual step, the old "walk over and ask" for things Copilot answers: formally done.

The announcement itself is worth scripting, because hedged announcements reopen the old path by tone alone: "Starting Monday, the report generates from the tracker and the old template is retired. I own this change; if something breaks, bring it to me, not back to the old way." Two sentences, a date, an owner, and a door that closes.

Retirement will surface real problems, and that's the design. Where the new way genuinely can't carry the load, the owner adjusts scope honestly rather than quietly reopening the old path for everything, because two paths is the stall condition. Expect mild discomfort and a few loud days. A team that feels nothing in week three retired nothing.

Week four: measure and decide

Rerun the week-one counts on the same sheet. Same workflows, same questions, now with the new way as the default. Then the owner writes a one-page verdict, and the format matters less than the discipline of the three findings: the numbers before and after, what the team says qualitatively, and the decision.

The decision is one of three, and all three are wins:

```
Day 31: the four exits

  EXPAND   it worked, numbers show it; next workflows      win
  FIX      real value, named blockers; adjust and rerun    win
  KILL     didn't repay the effort; verdict banked cheap   win
  DRIFT    no decision, full cost, no verdict         forbidden
```

Expand. It worked; the numbers show it. Extend to the next workflows, and let the one-pager travel upstairs and to neighboring teams. This is your recruiting document.

Fix. Partial success with identifiable blockers, often content quality (back to Chapter 7) or a workflow that needed different scoping. Adjust and run another focused cycle.

Kill. The chosen workflows didn't repay the effort. Retire the experiment honestly, document why, and either pick better workflows or bank the finding. A clean kill in thirty days is dramatically cheaper than an ambiguous fade over a year, and it preserves the credibility you'll want for the next attempt.

What's forbidden is the fourth outcome, the default one. Day 31 exists so it can't happen. Put the decision meeting on the calendar in week one, before anyone knows the answer, and the plan enforces itself.

> **WARNING: Trailing off undecided is the one forbidden outcome: full cost paid, no verdict collected.**

After the first thirty days

Expansion is the same plan pointed at more workflows or the next team, with one upgrade: now you have local proof, local champions, and prompts already tuned to your organization. Each cycle runs smoother than the last. Run two or three cycles and you'll notice something has shifted that no license could buy: the team has learned

how to adopt, which is a capability that outlives any particular tool, including this one.

The whole plan compresses to a checklist in Appendix B. Next chapter: the measurement layer in more depth, including what to report upward so the people who approve renewals see what you see.

CHAPTER 10

Measuring Whether It's Working

IN THIS CHAPTER

- Why activity numbers can't prove value
- The four metric families a budget review respects
- The honest costs column, and why volunteering it helps you
- The one-page scorecard format

Somewhere around month four, someone with budget authority will ask the only question that matters: is this worth what we're paying? This chapter makes sure you can answer with numbers instead of vibes, because in that meeting, vibes lose.

The good news is that Copilot measurement is simpler than the analytics industry wants you to believe. You need a small number of honest metrics, captured before and after, tied to workflows you deliberately changed. That's it. The rest of this chapter is what to count, what to ignore, and how to package it.

Ignore the activity numbers (mostly)

Microsoft's admin dashboards will happily report adoption statistics: active users, prompts submitted, features touched. These have exactly one legitimate use, and it's diagnostic: if licensed users aren't even signing in, you have a rollout problem worth investigating.

What activity numbers cannot do is prove value, because activity is what novelty produces for free. High usage of a tool that saves no time is a cost, not a win. Report activity as a footnote if you must; never let it headline.

> **WARNING: The moment 'engagement is up' is your main evidence, you've conceded you don't have real evidence.**

Count what the business already counts

The metrics that survive a budget review are the ones the business valued before Copilot existed. Four families cover nearly every case:

Hours recovered. The Monday report took four hours of assembly; it now takes forty minutes of review. The vacation re-entry cost half a day; now it's twenty minutes. Count time on the specific workflows you changed, using the same simple sheet from Chapter 9's baseline. Hours are the universal currency; every stakeholder converts them instinctively.

Cycle time. Proposal request to proposal sent. Question asked to question answered. Meeting held to actions assigned. Elapsed time shrinking on a process the business cares about is often more persuasive than effort saved, because customers and deadlines feel cycle time directly.

Quality and error signals. Fewer "wrong version" incidents after the Chapter 7 cleanup. Fewer dropped action items once recaps feed a confirmed task list. Rework rates on documents. Softer to capture, but one concrete example ("we caught a commitment in a recap that would

have been missed") carries real weight in a narrative.

Capacity redirected. The sharpest question in the family: what did people *do* with the recovered hours? "The coordinator now handles vendor escalations that used to wait days" turns saved time into visible business change. If you can name where even some of the hours went, the value argument closes itself.

One discipline across all four: **only claim workflows you actually changed.** Chapter 9's design gives you this for free, since you baselined specific workflows and retired specific steps. Resist inflating the count with vague org-wide productivity claims; a small, airtight number beats a large, assailable one in every meeting that matters.

The honest costs column

Credibility requires counting both sides, so your accounting includes: license costs for the users in scope, the training and check-in hours from the 30-day plan, the content cleanup investment, and any agent or consumption charges (Chapter 11's territory). Presenting costs yourself, unprompted, buys you the right to be believed on the benefits. It also usually flatters the result; a focused pilot's fully loaded cost against honest hours recovered tends to clear the bar without embellishment. If it doesn't clear the bar, you want to know that more than anyone, because Chapter 9 gave you the kill option for a reason.

The one-page scorecard

Package everything on a single page, updated monthly or quarterly. The layout I use:

1. **Workflows in scope**, one line each: what changed, who owns it.
2. **Before and after numbers** for each, same units, dates of capture.
3. **Costs**, fully loaded, same period.

4. **One story.** A single concrete moment (the caught commitment, the report that sent itself during the outage, the new hire who found the policy answer without asking anyone). Numbers earn the decision; the story makes it memorable.
5. **The ask or the verdict.** Expand, fix, kill, or renew, stated plainly.

Filled in, the whole page looks like this:

```
COPILOT VALUE SCORECARD, Q3 (example)

Workflows in scope
  1. Monday ops report        owner: ops coordinator
  2. Vendor email triage      owner: procurement lead

Before -> After (same units, capture dates 6/2 and 9/1)
  Report assembly     4.0 hrs/week  ->  0.7 hrs/week
  Triage to routing   1.5 days      ->  same day

Costs, fully loaded (quarter)
  14 licenses; 22 hours training and check-ins;
  content cleanup sprint (est. 30 hours, one-time)

One story
  8/14: recap surfaced a client commitment the room
  had missed; caught before the deadline, not after.

Verdict
  EXPAND to intake team. Renew licenses.
```

And when the page gets presented, the spoken version is one sentence, worth scripting like everything else in this part of the guide: "Fully loaded, this cost us [X] for the quarter and returned [Y] hours a month on the workflows we changed; my recommendation is [expand / renew / stop]." Everything else on the page is the appendix to that sentence.

Notice what this page is really doing: it's Chapter 8's three deficiencies, inverted into evidence. Named owners, changed workflows, operational numbers. Any executive can follow it in ninety seconds, and ninety seconds is what you'll get.

Measuring the unmeasurable, briefly

Some Copilot value resists clean numbers: better first drafts, faster orientation, the confidence of asking questions freely. Two honest tools exist for this layer. A short quarterly pulse survey (five questions, same five each time: does Copilot save you meaningful time in a typical week; would you give it up; what's your best current use; where has it wasted your time or misled you; which workflow should we point it at next) produces trendable sentiment, and the "would you give it up" question is startlingly clarifying. And collected stories, gathered in the Chapter 9 check-ins, build the qualitative file. Use both as seasoning, never as the meal. The meal is the before-and-after on workflows you changed on purpose.

That closes the core adoption arc: what it is, how to use it, what feeds it, how to roll it out, how to prove it. Part V walks the edges of the map, starting with what happens when the workflow you care about needs more than an assistant.

CHAPTER 11

Extend: Agents, Copilot Studio, and Automation

IN THIS CHAPTER

- Agents, Copilot Studio, and automation: what each block does
- The three-trait sweet spot where extension shines
- What the demo doesn't show: ownership, consumption costs, the complexity ceiling
- The five-question homework before anyone builds

Somewhere in your first months with Copilot, a sentence like this will be spoken: "This is great, but can it just *do* it automatically?" The report is easier to assemble, but someone still has to ask. The recap is excellent, but someone still turns it into tasks. The policy answers are right there, but new hires still ping the ops manager instead of asking Copilot.

That sentence marks the boundary between category two and category three from Chapter 2: between an assistant that helps a present, asking human, and something configured to do a defined job. Microsoft's answer to that boundary is the extend layer: agents, Copilot Studio, and the Power Platform automation family. This chapter covers what extension is genuinely good for, what it costs in ways the demo doesn't show, and the complexity ceiling where it stops being the cheap option.

What extension actually is

Three building blocks make up the extend layer, and they combine:

Agents are scoped assistants: grounded in specific content, given specific instructions, often published to a specific place. The HR policy agent that answers only from the approved policy library. The IT helpdesk agent in a Teams channel that resolves the top twenty questions and escalates the rest. The proposal assistant that knows your templates and past proposals and nothing else. Scoping is the entire point: a narrow agent grounded in curated content is more reliable than the general assistant precisely because its world is smaller.

Copilot Studio is where agents get built and configured: point-and-click for the straightforward cases, with room to add knowledge sources, actions, and behaviors. It's approachable enough that a capable business user can build a first agent, which is both its

promise and, as we'll see, its trap.

Automation (the Power Automate family) is the trigger-and-flow machinery: when a form is submitted, when a file arrives, when a date passes, do these steps. Automation is what removes the "someone still has to ask" limitation, and pairing it with AI steps is how work starts happening without a human initiating each instance.

Where extension genuinely shines

The honest sweet spot has three characteristics, and the best extension projects have all three:

The knowledge is curated and stable. Policy Q&A agents work because Chapter 7 taught you to build a clean, owned library for them to stand on. An agent grounded in chaos is just the grounding problem wearing a costume.

The process is simple and describable. Route the request, answer from these documents, collect these fields, post the summary, notify this person. If you can describe the whole job in a few sentences without saying "except when," extension will likely carry it.

The volume justifies configuration. The question answered forty times a month deserves an agent. The one answered four times a year deserves a document.

Classic wins, in rough order of how often I see them pay off: internal Q&A agents (HR, IT, operations policies), intake and routing (a form arrives, gets classified, lands with the right person with a summary attached), recap-to-task handoffs, and scheduled report drafting where Chapter 4's data legibility work has been done.

What the demo doesn't show

Agents are products, not settings. The click-to-build ease is real, and it obscures the operating reality: an agent that people rely on needs an

owner, testing, monitoring for wrong answers, and maintenance when its content changes, forever. Chapter 8's first deficiency applies with full force. Before anyone builds an agent, the question is not "can we" but "who owns this for its whole life?"

> **WARNING: An orphaned agent confidently serving stale answers is worse than no agent, because it wears your organization's authority.**

Consumption pricing needs adult supervision. Agent and automation capabilities often carry usage-based costs on top of per-user licensing, metered in ways that vary and change. The principle that survives the pricing churn: estimate volume before you build, check actual consumption in month one, and treat any agent with organization-wide reach as a budget line, not a rounding error. Verify current models against Microsoft's licensing pages; the specifics have the shelf life of fruit, the discipline doesn't.

The complexity ceiling is real. Here's the pattern to watch for. Extension projects start simple, then accrete: another exception, another condition, another system to touch. At some point the flow chart no longer fits on a screen, changes take a specialist a week, and nobody's sure what happens in the edge cases. You are now paying custom-application prices for flowchart-tool flexibility, and every future change costs more than it should. Chapter 1's purchase-order chain is the canonical case: a stripped-down version of it extends beautifully, and the real version, with validation rules and vendor exceptions, lives past this ceiling. The warning signs: more than a couple of systems involved, decision logic with real consequences, "except when" appearing more than twice in the description, or anyone saying "we'll just work around that limitation." Two or more of those, and it's time for the next chapter's conversation before you build, not after.

The governance sentence

Extension multiplies Chapter 13's concerns, so one sentence here as the placeholder: anything that acts automatically in your organization's name needs the same review a new employee's authority would get, meaning what it can access, what it can do unsupervised, and what must escalate to a human. Money, commitments, and personnel actions escalate. Always.

The homework before building

Compressed to a pre-flight, and the echo is deliberate: the first three questions are the sweet-spot traits from above, restated as go/no-go checks, because a trait you can admire and a question you must answer are different instruments. Four and five are the new ones, and they're the two the demo never mentions:

1. Is the knowledge it needs curated and owned? (If not, Chapter 7 first.)
2. Can you describe the whole job in five sentences without "except when"?
3. Does the volume justify it?
4. Who owns it for life, by name?
5. What's the consumption estimate, and who checks the actual in month one?

Five honest answers and extension will likely serve you well; it's a genuinely productive layer when scoped with discipline. But when question two keeps failing, when the workflow crosses systems and carries real decision logic, you've reached the edge of Microsoft's map. The next chapter is about the territory beyond it, and why crossing over costs far less than it did the last time you priced it.

CHAPTER 12

Build: When Copilot Isn't the Answer

IN THIS CHAPTER

- The three signs a workflow is beyond Copilot's map
- The full stay, extend, or build test
- What building costs now, and why your old math is wrong
- Buy vs. configure vs. build, and the advisor filter

Every chapter so far has worked inside Microsoft's map: the assistant, its apps, its content, its extension layer. This chapter is about the workflows that live beyond it, because your organization has some, and the most expensive mistake in the whole Copilot era is filing them under "solved because we have Copilot."

They are not solved. They are unaddressed, with a license fee attached. The second most expensive mistake is assuming that addressing them requires a development team and a six-figure budget, because that stopped being true, and most leaders are still carrying 2019 prices in their heads.

The three signs you're in build territory

A workflow belongs beyond the assistant-and-extension world when it has one or more of these characteristics. You met them briefly in Chapter 2; here they are with their full weight.

It crosses systems. The purchase-order chain from Chapter 1 is back, and this is where it finally gets its answer: it arrives by email, gets validated against the ERP, updates the tracking sheet, and notifies the channel. The work *is* the chain, and no assistant inside any single app

can own a chain. Extension can sometimes bridge two friendly systems; when the chain is the whole job and the systems weren't built to talk, you're building.

It runs on your data, in your shapes. Work orders, SKUs, inspection results, client records, with rules particular to your business. Copilot can read a document describing your rules. It cannot *be* the system that stores your records and enforces your rules consistently, because that's what applications are, and consistency is the entire job.

It carries decision logic with consequences. Approve, escalate, reorder, flag. Logic that must be explicit, testable, auditable, and identical on Friday afternoon and Monday morning. Chapter 1 said it and it's the load-bearing sentence of this chapter: that's an application property, not an assistant property.

The stay, extend, or build test

The whole guide's decision framework in its final form. For any workflow, ask in order:

Could one person do this task entirely inside Microsoft 365 content, with a human present every time it happens? Yes: **stay**. Train the patterns, fix the content, take the win.

Is it mostly M365 work that needs a connection, a trigger, or a scoped Q&A agent, describable in five sentences without "except when"? Yes: **extend**, with Chapter 11's homework done.

Does it cross systems, run on your own structured records, or carry consequential decision logic? Then **build**, and no amount of additional Copilot licensing will change the answer. Naming this boundary honestly is how you stop paying assistant prices for application problems.

Appendix C turns this into a worksheet, because the test works best on paper, applied to a real workflow, in a meeting where someone was

about to say "can't Copilot do that?"

What building looks like now

Here's the part that changed while everyone was watching the assistant wars, and it's the reason this chapter exists in a guide for business readers rather than developers.

Building a focused business application used to mean a project: developers, months, six figures, and a maintenance contract. That's why the workflows in this chapter stayed manual for a decade; the math never worked for a problem that wastes ten hours a week. The current generation of AI-powered development tools rewrote the math. Small teams, sometimes a single technically-comfortable person, now ship real applications in weeks: describing the product in plain language to an AI app builder, getting a working version in days, then hardening it with AI coding tools under human review. The AI features themselves (the document extraction, the classification, the drafting) ride on the same class of models that power Copilot, called through APIs inside your own application, doing exactly what you specify.

I run my own product development this way, and I build this class of applications for clients, so let me be honest in both directions. The tools are genuinely that fast for focused, well-scoped applications: an intake processor, a triage system, a reporting engine, a knowledge tool. They are not magic for sprawling, everything-connected platforms, and speed of building has never been the hard part anyway. The hard parts remain what this guide has said all along: choosing the right workflow, capturing the baseline, owning the change, and retiring the old way. A custom application dropped into the three deficiencies (no owner, no changed process, no before-number) stalls exactly like a Copilot rollout does. Chapters 8 through 10 apply in full; only the tool changed.

What this means for your math: workflows you priced out of consideration years ago deserve repricing. The question "what would it cost to actually fix our intake process?" has a different answer now, often by an order of magnitude, and the leaders who know that are quietly compounding advantages over the ones who don't.

Buy, configure, or build, and who to trust

One more layer of honesty, because build territory has its own decision. Some of these workflows are served by off-the-shelf products (buy), some by extension-style configuration of platforms you own (configure), and some genuinely warrant a custom build. The tiebreakers: buy when your workflow is standard enough that a product fits without contortions, configure when Chapter 11's sweet spot holds, build when the workflow *is* your way of operating and bending it to a product's assumptions would cost the advantage.

And whoever helps you decide, apply one filter. The Copilot reseller for whom everything is an agent, the dev shop for whom everything is custom, the vendor for whom everything is their product. The test from this chapter is valuable precisely because it produces different answers for different workflows, and an advisor worth paying will sometimes tell you to stay exactly where you are.

> **WARNING: Be suspicious of any advisor whose answer never varies.**

You now have the full map: the assistant, its edges, and the territory beyond. One chapter remains, and it's the one that keeps all of this out of trouble.

CHAPTER 13

Governance, Security, and Not Getting Fired

IN THIS CHAPTER

- The five preventable incidents, and the principle behind each
- The verify-at-source norm as policy
- Shadow AI as a supply problem
- The seven-line AI policy that fits on a page

Every previous chapter has been about getting value from AI at work. This one is about doing it without creating the incident that makes your organization swear off AI for three years, or the one with your name in the postmortem. The good news: staying safe requires a small number of durable principles, not a compliance department. The incidents that matter cluster into five preventable patterns, so this chapter is organized around preventing exactly those five.

Incident one: the overshared file in the answer

The scenario: someone asks Copilot an innocent question and receives content they were never meant to see, cited politely, because it was technically visible to them all along. Chapter 7 covered the fix in full, so here it earns only its governance framing: **this is the single most likely way Copilot embarrasses your organization, and it's preventable in advance.** The oversharing audit is not an IT nicety; it's the pre-launch safety check. If your rollout is ahead of your permissions review, your rollout is ahead of itself.

The durable principle: Copilot doesn't create access, it reveals it. Govern access and you've governed most of what Copilot can surface.

Incident two: the confident wrong answer, acted on

The scenario: Copilot states a number, a policy, or a commitment with total assurance; a human acts on it; the number was stale or subtly wrong. No villain, just Chapter 6's mechanism (retrieval finds relevant, not correct) meeting Chapter 3's warning (confidence is not accuracy) at the worst moment.

The governance response is a norm, stated plainly and repeated until it's culture: **AI output that feeds a decision gets verified at the source.** Where Copilot shows its sources, clicking through is the whole discipline; where it doesn't, the source gets found the old way. Where the stakes are money, legal exposure, or someone's employment, verification isn't optional courtesy, it's the process. Teams internalize this fastest when leaders model it out loud: "Copilot says the renewal is March 15, let me confirm in the contract before we plan around it."

The durable principle: the human who acts owns the action. Chapter 1's assistant model was a governance policy all along.

Incident three: the data that left the building

The scenario: an employee, quite reasonably trying to be productive, pastes client data or internal financials into a personal AI chatbot, outside your organization's protections. Nothing malicious occurred, and your data now lives somewhere your agreements don't reach.

Three moves prevent this one:

Give people the sanctioned path. The most effective control is a good option: if the work-grounded Copilot and the commercially protected chat are available and decent, the temptation to wander to personal tools collapses. Organizations that ban AI without providing it don't stop AI use; they just stop being able to see it. Shadow AI is a

supply problem before it's a discipline problem.

Make the boundary teachable in one sentence: work content goes only into tools your organization provides, signed in with your work account. That sentence covers the paste, the upload, and the clever workaround, and a new hire can hold it.

Know your data protections. The business versions of Microsoft's AI tools generally commit that your prompts and content aren't used to train foundation models and stay within your organization's boundary. Verify the current terms rather than assuming, and make sure whoever answers employee questions can state them plainly, because "is this thing reading our email?" deserves a real answer, not a shrug.

Incident four: the meeting that shouldn't have been transcribed

The scenario: recording and transcription, the fuel for Chapter 3's best features, capture a conversation that legal, HR, or basic human decency says shouldn't have been captured, or capture people who didn't know they were being recorded.

The governance here is judgment plus defaults. Defaults: recording notifications on (they are, by design; don't defeat them), and clear team norms about which meetings record. Judgment: some conversations (personnel matters, legal discussions, sensitive negotiations, anything where candor matters more than recall) should be explicitly unrecorded, and anyone in the meeting should feel licensed to say so. Also know your ground rules: recording-consent law varies by jurisdiction, and multi-state or international meetings inherit the strictest rule in the room. Your legal counsel sets that policy; your job is making sure meeting organizers know it exists.

The durable principle: transcription turns conversation into a document, and documents are discoverable, forwardable, and permanent.

> **WARNING: Record like everyone will eventually read it, because they might.**

Incident five: the agent nobody was watching

The scenario, previewed in Chapter 11: an agent or automation, built in an enthusiastic afternoon, keeps operating after its content went stale or its builder left, confidently serving wrong answers or acting beyond what anyone remembers authorizing, wearing your organization's name the whole time.

Governance for the extend-and-build layer fits in four requirements, the same four Chapter 11 planted, now stated as policy and applied to anything that acts automatically: **an owner by name for its whole life; a written scope of what it may access and do unsupervised; escalation rules (money, commitments, and personnel actions always route to a human); and a review date on a calendar.** An agent meeting those four requirements is an asset. An agent missing any of them is an incident with a launch date.

The one-page AI policy

Most organizations need less policy than they fear and more clarity than they have. If yours has nothing, these seven lines, adapted to your reality and blessed by whoever must bless things, cover the five incidents:

1. Use the AI tools the organization provides, signed in with your work account; work content goes nowhere else.
2. You own what you send, ship, and decide. AI output is a draft until a human verifies it.
3. Anything feeding a decision gets checked at the source, every time.

4. Sensitive conversations don't get recorded; anyone can invoke this.
5. Access is governed deliberately: new shares start restrictive, and the oversharing review runs on a schedule.
6. Anything that acts automatically has a named owner, a written scope, escalation rules, and a review date.
7. When unsure, ask before, not after. No one gets in trouble for the question.

Print it, teach it in the Chapter 9 rollout, and revisit it quarterly. Governance that fits on a page gets followed; governance that fits in a binder gets signed and forgotten.

The closing thought

Thirteen chapters ago, a license showed up on a Tuesday. Between then and now, the argument of this guide has stayed constant even as the subject moved from prompts to permissions to pilots to policy: the technology is the entry fee, and the value comes from what people do differently, deliberately, with someone's name on it.

That's also why none of this expires. Microsoft will rename products, move buttons, and ship capabilities this guide couldn't have described. The grounding question will still sort them. The three deficiencies will still stall them. The 30-day plan will still prove them. The stay, extend, or build test will still route them. And the person in your organization who can run those four moves calmly, while everyone else oscillates between magic and uselessness, becomes something more valuable than a Copilot expert. They become the one who knows how to make change stick.

That might as well be you. You've read the guide.

APPENDIX A

The Five Prompt Patterns (One-Page Card)

*Every effective request mixes four ingredients: **Goal** (what you want), **Context** (what only you know), **Constraints** (length, tone, exclusions), **Source** (what content to work from). When output disappoints, one of these is missing.*

1. Catch Me Up

"Summarize [this thread / this channel since Monday / this meeting]. What was decided, what's still open, and what needs something from me?"

Use for: threads, meetings you missed, vacation re-entry, weekly review.

2. Draft From Intent

"Draft [thing] for [audience]. Goal: [outcome]. Context: [facts only you know]. Tone: [tone]. Keep it [length]."

Use for: emails, updates, proposals, posts. The Context line is what makes it sound like you.

3. Transform

"Turn [this] into [that], keeping [what matters]. Put anything unclear in a questions list."

Use for: notes to tables, documents to summaries, narratives to decks, chaos to agendas.

4. Interrogate

"Based on [source]: [question]." Advanced: "What's missing? What would a [skeptical reader / lawyer / frustrated customer / CFO] challenge?"

Use for: contracts, reports, your own drafts before they ship.

5. Coach Me

"Review this [thing]. How will [audience] receive it? What would you strengthen, soften, or cut?"

Use for: anything delicate, fifteen seconds before the point of no return.

Two habits: never send unreviewed output, and treat the first answer as a first answer. Nudge, refine, accept.

APPENDIX B

The 30-Day Adoption Plan Checklist

Before day one

- ■ Owner named, with authority over the affected workflows
- ■ Two or three specific workflows chosen (frequent, annoying, measurable, low blast radius)
- ■ Worst content risks addressed or scheduled (oversharing audit, stale critical docs)
- ■ Day-31 decision meeting on the calendar now

Week one: baseline and setup

- ■ Before-numbers captured per workflow (hours, cycle time, interruptions) on a shared sheet
- ■ Expectation conversation held (assistant not oracle; drafts not finals; verify facts; share wins)
- ■ One training session: five patterns demonstrated on the team's actual workflows
- ■ Pattern card distributed; each person picks two moves to drill

Week two: run both paths

- ■ Daily use on chosen workflows, old path still available
- ■ Two ten-minute check-ins: wins shared specifically, flops diagnosed via the four ingredients
- ■ Retirement list drafted: what stops next week, exactly

Week three: retire the old way

- Retirement announced with a date and an owner (the scripted two sentences from Chapter 9)
- Old steps formally stopped (the meeting, the manual assembly, the walk-and-ask default)
- Problems surfaced and scoped honestly; no quiet reopening of the old path
- Owner visibly using the new way

Week four: measure and decide

- After-numbers captured, same sheet, same units
- One-page verdict written: numbers, team sentiment, one story
- Decision made and recorded: **Expand / Fix / Kill** (DRIFT is not an exit)
- If expand: next workflows or next team chosen; one-pager shared upstairs

APPENDIX C

The Stay, Extend, or Build Worksheet

Run one workflow at a time through these questions, in order, in writing.

The workflow: _____

What it costs today (hours/week, cycle time, or interruptions):

_____

Question 1. Could one person do this task entirely inside Microsoft 365 content, with a human present every time it happens?

→ **Yes: STAY.** Actions: train the relevant patterns (Ch. 5), fix the content it depends on (Ch. 7), baseline and measure (Ch. 9–10). Stop here.

→ No: continue.

Question 2. Is it mostly M365 work that needs a connection, a trigger, or a scoped Q&A agent, describable in five sentences without "except when"?

→ **Yes: EXTEND.** Before building, complete the homework: knowledge curated and owned? Volume justifies it? Lifetime owner named? Consumption estimated, month-one check assigned? Escalation rules written (money, commitments, personnel always route to a human)?

→ No, or "except when" keeps appearing: continue.

Question 3. Does it cross systems, run on your own structured records, or carry consequential decision logic?

→ **Yes: BUILD territory.** Next decision: **Buy** (a product fits without contortions) / **Configure** (the extend sweet spot actually holds) / **Build** (the workflow is your way of operating). Reprice before assuming; the math changed.

Warning signs you chose Extend but belong in Build: more than two systems involved; the flow chart no longer fits on a screen; consequential decisions embedded in the flow; "we'll work around that limitation."

Whatever the answer: the adoption rules still apply. Owner by name, baseline before, something old retired, decision on day 31.

ABOUT THE AUTHOR

About the Author

Rosemarie Withee is the founder of Portal Integrators, a Seattle consultancy that builds AI applications for operations teams and helps organizations make new technology actually stick. She is the author of six Wiley books on Microsoft 365 technologies, including Microsoft Teams and SharePoint titles, and has spent more than thirteen years working in the gap between the software organizations buy and the value they get from it.

She writes about AI adoption, Microsoft 365, and building software at portalintegrators.com/writing.

If this guide was useful and you want help applying it, Portal Integrators works with operations teams on exactly what these chapters describe: choosing the first workflow, running the 30-day plan, and building the applications Copilot can't be. The first conversation is a working session, not a pitch.

The license shows up on a Tuesday.

Whether it changes how your team works has surprisingly little to do with the software. Copilot at Work is the practical guide for the part that actually decides the outcome: what the assistant is genuinely for, why two companies with identical licenses get wildly different results, and how to run an adoption that ends in a verdict instead of a shrug.

Inside:

- The one-sentence definition of Copilot, and the four boundaries in it
- Five prompt patterns that cover most office work
- The content cleanup that upgrades every answer for every user
- A 30-day team plan with a decision on day 31: expand, fix, or kill
- The stay, extend, or build test for when Copilot isn't the answer
- A seven-line AI policy that fits on a page

The technology is the entry fee. The value is what your team does differently, deliberately, with someone's name on it.

Rosemarie Withee

is the founder of Portal Integrators and the author of six Wiley books on Microsoft 365. She builds AI applications for operations teams and helps organizations make new technology stick.

This guide is free. Share it with your team.

Help running the 30-day plan: portalintegrators.com