GITHUB COPILOT ONBOARDING · PLAYBOOK

# Structured onboarding decides whether GitHub Copilot returns value or stalls at 23%.

Companies do not fail at GitHub Copilot because the tool is weak. They fail because literacy is unequal. This playbook turns a license into a program: twenty-four SDLC personas across seven clusters, a five-stage journey from preparation to autonomy, a quick win inside the first seventy-two hours, and a Center of Excellence that owns the cadence after the formal program ends. Teams with formal onboarding reach 80 percent active adoption in ninety days; teams without it settle near 23 percent.


AUTHOR

Paula Silva

ROLE

AI-Native Software Engineer

CONTACT

in/paulanunes


READING TIME

~ 45 minutes, end to end

| 24 | 7 | 5 |
|---|---|---|
| SDLC PERSONAS | PERSONA CLUSTERS | ONBOARDING STAGES |

80%

ACTIVE ADOPTION TARGET

CHAPTERS

Chapter 00

## Why structured onboarding decides the return

Why GitHub Copilot does not return value when literacy is unequal, what the evidence says about active adoption, the seven pitfalls that stall most rollouts, and the six principles the program is built on.

Foundations →

Chapter 01

## Twenty-four SDLC personas across seven clusters

The twenty-four SDLC roles grouped into seven clusters, the pain each carries today and the gain agents bring, the three-plane tool matrix, and one vocabulary per cluster reporting into one shared dashboard.

Personas →

Chapter 02

## A five-stage program from preparation to autonomy

The five stages from a preparation gate to autonomy, each with one explicit exit criterion, resting on a four-layer enablement stack of personas, stages, labs, and measurement.

Framework →

Chapter 03

## Literacy tracks by cluster and Copilot tool

How each cluster maps to a GitHub Copilot surface: agent mode for engineering, repository instructions and MCP for context, the model picker as a cost decision, and tracks for platform, data, product, and security.

Literacy →

Chapter 04

## Quick wins in seventy-two hours, measured every fortnight

One concrete quick win per cluster inside the seventy-two hour window, pattern cards and a shared prompt library, six KPIs from reach to risk, and cost governance with GitHub AI Credits.

Metrics →

Chapter 05

## Sustain adoption with change management and a Center of Excellence

The resistance map and a four-step conversion, Kotter applied to agentic adoption, the Agents Center of Excellence and community of practice, and the six-month roadmap with Monday-morning actions.

Adoption →

READING TIPS

> **KEYBOARD SHORTCUTS**
>
> Press `/` to search. Use `j` and `k` to move between chapters. Press `t` to toggle theme. Press `g` to go to top.

---

**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

00 THE LITERACY GAP

# Why structured onboarding decides the return.

Companies do not fail at GitHub Copilot because the tool is weak. They fail because literacy is unequal. Licenses are bought, seats are provisioned, and adoption still stalls: senior engineers run with the tool, QA never opens it, the product manager sees no connection to their work, and the CISO has open questions. The difference between paying for AI and getting value from it is the program wrapped around the tool. This chapter measures that gap, states the evidence for active adoption, names the seven pitfalls that stall most rollouts, and sets out the six principles the program is built on.

## The literacy gap no one budgets for

Organizations buy GitHub Copilot licenses and expect adoption to follow. It does not. According to GitHub Octoverse 2025, teams with formal onboarding reach 80% active adoption within 90 days. Teams without it reach 23%. That is a 3.5x gap. The differentiator is not budget, not seniority, not the tech stack. It is the program built around the tool. The gap has a name that no line item captures: unequal literacy. Senior engineers run with the tool, QA never opens it, the product manager sees no connection to their work, and the CISO has open questions. The license is uniform. The literacy is not.

The default rollout is a license and a welcome email. Active adoption peaks around week two, then collapses to roughly 12% by day 60 (Forrester 2025). The billing model makes that waste legible. GitHub Copilot moved to usage-based billing on June 1, 2026, priced in GitHub AI Credits, where one credit equals one US cent. Agent mode runs and premium requests draw down those credits. An organization that provisions the seats but never builds literacy pays for the credits and captures almost none of the return. Onboarding gets treated as an event. It has to run as a program.

# What the evidence says about active adoption

The evidence is consistent across sources. Formal programs reach 80% active adoption at 90 days; informal rollouts settle near 23% (GitHub Octoverse 2025). Each dollar invested in structured training returns USD 4.20; self-taught training returns USD 1.80. MIT Sloan 2024 found that framing the program as augmentation, not automation, produces 2.8x the engagement. The number that pays for the program is not the seat count. It is the active-user rate, the share of licensed users active at least five days a week.

Depth matters as much as reach. Active adoption is defined as at least one accepted GitHub Copilot suggestion per workday over the trailing two weeks. That definition separates seat holders from practitioners. A developer who runs one multi-step agent mode session per week is using the platform differently from one who accepts an autocomplete and moves on. The program teaches the second behavior: choosing the right model in the Copilot picker for the task, Claude Fable 5 or Opus 4.8 for hard reasoning, Sonnet 5 or Haiku 4.5 for fast edits, Gemini 3.1 Pro or a GPT-5.x model when the task fits, and writing the agent mode prompt that turns a backlog item into a pull request. Reach without depth is a vanity metric.

# Seven pitfalls that stall most rollouts

Field evidence from the McKinsey Global Survey on AI 2024 and Forrester AI Adoption Pitfalls 2025 points to a short list of recurring failure modes. One-shot training with no follow-up: adoption falls to 12% within 60 days. Engineer-only programs: adoption silos, and roughly half the potential return goes uncaptured because QA, product, and security never begin. No real use cases: generic tutorials kill motivation in week one. No clear AI usage policy: the CISO blocks the rollout and teams freeze.

The remaining three are quieter and just as fatal. No internal champions: practice never diffuses socially, so it stays trapped in the early-adopter pocket. No metrics, or the wrong ones: the budget gets cut at the first quarterly review because nobody can show the return. One curriculum for everyone: high dropout and low relevance, because a product manager and a backend engineer do not learn the same tool the

same way. Every one of the seven traces back to a single error, treating a deployment event as a finished program.

## Six principles the program is built on

The program rests on six principles the research validates as determinants of success. Persona-first: each role has its own motivations, fears, vocabulary, and work context, so a generic guide to GitHub Copilot does not land for a product manager, and literacy has to speak each persona's language. Quick wins first: habits change on positive reinforcement, so the program engineers a visible win inside the first 72 hours before it introduces advanced concepts. Social learning: Rogers, in Diffusion of Innovations, shows adoption accelerates when people watch peers succeed, not only experts, which is why internal communities of practice beat top-down training.

The other three govern trust and durability. Psychological safety: people do not experiment with a new tool when they fear looking incompetent, so the environment has to be explicitly safe for trial and error. Visible governance: adoption rises when the rules are clear, what the agent may and may not do, who owns what, how data is protected, and that clarity matters most for non-technical personas and leaders. Continuous measurement: without metrics the program becomes a black box, so adoption, quality, and satisfaction get measured every fortnight and the program adjusts. In the autonomy stage these principles harden into artifacts: a shared .github/copilot-instructions.md that encodes conventions once, and MCP servers that give agent mode governed access to internal context.

> DEFINITION
>
> Six principles, one rule: onboarding is not an event, it is a platform. Persona-first over generic. Quick wins over theory. Social learning over top-down. Psychological safety over fear. Visible governance over ambiguity. Continuous measurement over faith.

# References

Primary sources for the adoption gap, the empirical evidence, the seven pitfalls, and the principles the program is built on.


- GitHub Octoverse 2025, the adoption gap: 80% active adoption with formal onboarding versus 23% without.
- Forrester, AI Adoption Pitfalls, adoption collapses to roughly 12% within 60 days without a sustained program.
- MIT Sloan Management Review, AI in the Workplace, augmentation framing produces 2.8x the engagement of automation framing.
- McKinsey Global Survey on AI, field evidence for the recurring adoption failure modes.
- Everett M. Rogers, Diffusion of Innovations, 5th Edition, the social-learning basis for communities of practice over top-down training.
- GitHub Copilot Documentation, the reference platform for agent mode, the model picker, and .github/copilot-instructions.md.
- Usage-based billing and GitHub AI Credits, the credit model that makes stalled adoption a visible cost.


---


**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes



v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

01 ROLES AND CLUSTERS

# Twenty-four SDLC personas across seven clusters.

The organizations that get value from agentic AI onboard the whole delivery organization, not just their senior developers. This chapter maps that organization: twenty-four SDLC roles grouped into seven clusters, each with its own pain today, its own agent gain, its own three tool planes, and its own vocabulary. The clusters differ in almost everything except the dashboards they report into. That is the design. Speak each cluster's language, measure them all on one instrument panel.

## Seven clusters, twenty-four roles

The SDLC is not a single role, and any program that treats it as one fails the moment it reaches someone who is not a senior developer. This chapter maps the whole delivery organization: twenty-four roles, grouped into seven clusters. The grouping draws on three references that delivery organizations already use to describe themselves, the DORA 2024 State of DevOps report, SAFe 6.0, and IEEE SWEBOK v4. The seven clusters are Engineering with six roles, Quality and Reliability with three, Platform and Infrastructure with four, Product and Design with three, Data and AI with three, Security and Compliance with two, and Leadership and Strategy with three.

The cluster, not the individual, is the unit of onboarding. Two backend engineers and a mobile engineer share a primary tool, a review workflow, and a vocabulary, so they train together and measure together. A QA engineer and a product manager do not, so putting them in the same room produces a demo that lands for neither. Clustering keeps every session grounded in one cluster's real work: the artifacts on screen come from that cluster's current sprint, and the conventions the cluster already follows go into a shared .github/copilot-instructions.md file that every agent session in that cluster reads before it writes a line.

# Per-persona pain today and the agent gain

Each cluster carries a specific pain that predates any agent. Engineering loses hours to boilerplate and repetitive refactoring, and full stack engineers lose more to context switching between layers. Quality and Reliability writes test scaffolding by hand and diagnoses incidents from scattered logs. Platform and Infrastructure repeats configuration and chases drift across many systems. Product and Design synthesizes feedback and requirements manually. Data and AI debugs pipelines and waits through long experiment cycles. Security and Compliance triages alerts and false positives and maps controls to policy by hand. Leadership and Strategy has no clean read on productivity or tool ROI.

### From author to reviewer

The gain has the same shape in every cluster: the mechanical work moves to the agent, and the judgment stays with the person. Engineering uses GitHub Copilot in agent mode to take a low-complexity backlog item from branch to pull request without leaving the editor. Quality generates a full unit-test suite for an uncovered class in about fifteen minutes against three to four hours by hand, and teams that generate tests from requirements report 60 to 70 percent less time writing them. Platform describes a Bicep or Terraform module and reviews the generated code. Product builds a Copilot Studio agent that answers backlog questions from a SharePoint sheet, and PMs report recovering four to six hours a week. Security triages the top findings with GitHub Advanced Security and Copilot Autofix. Leadership reads DORA metrics instead of anecdotes.

The common thread is a change of role. The platform engineer stops writing Terraform from scratch and starts reviewing generated code. The QA engineer stops writing scaffolding and starts deciding which of a hundred generated cases actually matter. Formally trained developers produce 55% higher code quality than self-taught peers on the same tool, which is the measurable distance between accepting suggestions and directing them.

# The tool matrix of primary, secondary, and governance planes

Every cluster gets three tool planes, not one. The primary plane is where the cluster does its daily work. The secondary plane extends it. The governance plane keeps the work auditable. Engineering runs GitHub Copilot in agent mode as primary, GitHub Actions as secondary, and GitHub Advanced Security as governance. Quality runs GitHub Copilot for tests, Azure Monitor with Application Insights as secondary, and GitHub Actions in CI as governance. Platform runs GitHub Copilot for infrastructure as code, the Azure Developer CLI as secondary, and Defender for Cloud as governance. Product runs Copilot Studio, Microsoft 365 Copilot as secondary, and Purview over research data as governance. Data and AI runs Azure AI Foundry, Microsoft Fabric as secondary, and Purview as the catalog. Security runs GitHub Advanced Security, Defender for Cloud as secondary, and Purview with Compliance Manager as governance. Leadership runs Microsoft 365 Copilot, GitHub Insights with DORA metrics as secondary, and Azure AI Foundry for governance.

The three planes share one billing surface and one model surface. Agent-mode work bills as GitHub AI Credits, where one credit equals one cent, under the usage-based billing that took effect on June 1, 2026, so the depth metric and the invoice read from the same signal. The model behind every session comes from the Copilot picker: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro, and the GPT-5.x family, so a cluster matches a model to a task without leaving the tool. At the autonomy stage, clusters extend the primary plane with Copilot Extensions built on the Model Context Protocol, which is how a cluster's own systems become first-class context for its agents.

> DEFINITION
>
> Three planes per cluster: primary is where the daily work happens, secondary extends it, governance keeps it auditable. All three bill as GitHub AI Credits and draw their model from the Copilot picker, so depth and cost read from one signal.

## One vocabulary per cluster, one set of dashboards

Different clusters speak different languages, and the program honors that. A demo for engineers talks about branches, pull requests, and refactoring. A demo for product managers talks about backlog, feedback, and roadmap. A demo for security talks about findings, autofix, and audit trails. The same tool shown in the wrong vocabulary reads as someone else's tool, which is the most common reason a cluster never opens it. Framing matters here too. MIT Sloan finds that augmentation messaging produces 2.8 times the engagement of automation messaging, and augmentation is easiest to hear in your own vocabulary.

The vocabulary diverges, the dashboards converge. Every cluster reports into the same instrument panel, so leadership reads one signal across the whole organization. Reach is the active-user rate. Depth is agentic actions per week, the count of multi-step agent-mode sessions that separates autocomplete from orchestration. Throughput is the DORA quartet of lead time, deployment frequency, change failure rate, and time to restore. Risk is GitHub Advanced Security findings fixed and merged. One vocabulary per cluster keeps adoption real; one set of dashboards keeps it fundable.

> DEFINITION
>
> One cluster, one primary tool, one vocabulary, one row on the shared dashboard. The twenty-four roles never train as twenty-four separate curricula and never report as twenty-four separate metrics. Seven clusters is the unit small enough to speak each language and large enough to measure as one program.

## References

Primary sources for the seven-cluster map, the per-cluster tool matrix, and the shared dashboard.

- DORA, 2024 State of DevOps Report, reference for mapping delivery roles across the seven clusters.
- Scaled Agile Framework, SAFe 6.0, the role taxonomy behind the cluster map.
- IEEE SWEBOK v4, Software Engineering Body of Knowledge, the reference for the role profiles in each cluster.
- GitHub Copilot Documentation, the primary plane for the Engineering, Quality, and Platform clusters.
- GitHub Advanced Security, the governance plane for Engineering and the primary plane for Security and Compliance.
- Microsoft Copilot Studio Documentation, the primary plane for the Product and Design cluster.
- Azure AI Foundry Documentation, the primary plane for the Data and AI cluster.
- DORA, the four key metrics, the throughput signal on the shared dashboard.

---

**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

02 FIVE STAGES

# A five-stage program from preparation to autonomy.

A GitHub Copilot license does not create fluency. A program does. This chapter lays out the five stages the program runs across twenty-four weeks: a preparation gate that has to close before anyone starts, the augmentation framing and quick wins that build the first habit, and the operational fluency and autonomy that let the formal program retire. Each stage carries one explicit exit criterion, so progress is measured by evidence rather than by the calendar. The whole sequence rests on a four-layer enablement stack, and skipping any layer breaks the one above it.

## Stage 0: preparation and the prerequisites gate

Stage 0 runs from week minus two to week zero, and it is a gate, not a warm-up. Nine prerequisites have to be in place before a single persona sees a demo, with no exceptions. Publish the AI usage policy in writing. Provision GitHub Copilot Enterprise with agent mode enabled, and define a .github/copilot-instructions.md file per repository so every agent session reads the same conventions. Provision the adjacent workspaces the clusters will need: Azure AI Foundry, Copilot Studio, GitHub Advanced Security, Defender for Cloud, Microsoft Fabric, and Purview. Identify two champions per team and put them through two weeks of advanced training before the broad kickoff. Define the program KPIs in writing so the first report has a baseline to measure against. The exit criterion is concrete: policy published, champions trained, sandboxes ready.

The gate matters because the three most common ways a rollout stalls all trace back to a prerequisite that was skipped. Without a written policy, the CISO blocks the rollout and teams freeze. Without trained champions, there is no one to answer the first hard question, and adoption never diffuses past the early volunteers. Without provisioned sandboxes, the first quick win slips past its window and the habit never forms. Budget belongs in the gate too. Since June 1, 2026, GitHub Copilot bills

premium usage on GitHub AI Credits, where one credit is one US cent, so the program owner sizes the credit allocation before the pilot rather than reacting to the first invoice. The prerequisites gate is not bureaucracy. It is the cheapest insurance the program buys.

## Stages 1 and 2: augmentation framing and quick wins

Stage 1 is awareness, weeks one and two, and its whole job is framing. Present the program as augmentation, not automation. The distinction is not cosmetic: MIT Sloan reports that an augmentation framing produces about 2.8 times the engagement of an automation framing, because people lean into a tool that makes them better and pull away from one they read as a replacement. Run live demos of sixty to ninety minutes, one per cluster, and build each demo on a real artifact pulled from the current sprint. Generic tutorials fail here, every time, because a QA engineer watching a backend refactor sees nothing that maps to the work in front of them. The exit criterion is participation with relevance: every persona attended a demo that used their own backlog, not a sample repository.

Stage 2 is quick wins, weeks three to six, and it is timed to the habit window. The first measurable success has to land inside 72 hours, the threshold below which a new behavior tends to stick and above which it tends to lapse. Each cluster gets one concrete win. Engineering takes a low-complexity backlog item from branch to pull request in agent mode without leaving the editor. QA generates a full unit-test suite for an uncovered class, which typically takes about 15 minutes against 3 to 4 hours by hand. Platform generates a GitHub Actions pipeline for a new service, using the project configuration as context. Product builds a Copilot Studio agent that answers backlog questions from a SharePoint sheet. Security triages the top three CVE findings with GHAS Security Autofix. Data generates an exploratory notebook in about 15 minutes. The model picker is part of the craft: route high-volume, low-stakes tasks to Haiku 4.5 to conserve credits, and reserve Opus 4.8 or Claude Fable 5 for the reasoning-heavy work. Every win is captured as a one-page pattern card, the seed of the prompt and recipe library. The exit criterion: every persona logged at least one quick-win artifact.

## Stages 3 and 4: operational fluency and autonomy

Stage 3 is operational fluency, weeks seven to fourteen, where occasional use becomes daily practice. The mechanics are hands-on: pair programming sessions, prompt dojos where teams refine and compare prompts against a shared task, and a monthly showcase where personas present what they shipped. This is where the .github/copilot-instructions.md file earns its keep. A convention a team argues out in a dojo, how it handles exceptions, how it names tests, how it structures a service, becomes standing context that every future agent session reads without being re-explained. The prompt library stops being a folder of snippets and turns into a curated asset. The exit criterion has three parts: 60% active weekly use, a curated prompt library, and a showcase running on a fixed cadence.

Stage 4 is autonomy, weeks fifteen to twenty-four, and it is where the program plans its own exit. Teams build custom agents for their recurring workflows and extend agent mode with MCP servers that expose internal systems, an issue tracker, a deployment API, a documentation store, as first-class context the agent can query directly. Model selection becomes deliberate rather than default: teams choose from the Copilot picker by task, matching Sonnet 5 or Gemini 3.1 Pro to the shape of the work and the credit budget. OKRs start to carry AI metrics as a normal line, not a novelty. The exit criterion is the clearest signal in the whole program: the champions retire the formal structure and a community of practice takes over the cadence. The autonomy stage succeeds when the program can remove itself.

## Exit criteria and the four-layer enablement stack

The five exit criteria are the spine of the program, and they are cumulative. A team does not advance because six weeks passed; it advances because it met the criterion, an attended demo, a logged artifact, a weekly-use threshold. Gates, not calendars, decide progress, which is what keeps a stage from being declared done while half the personas are still stuck. When a stage stalls, the criterion tells you exactly which layer to reinforce before moving on.

### The four-layer enablement stack

Underneath the stages sits a four-layer enablement stack, and each layer depends on the one below it. Layer one is personas: the 24 SDLC roles grouped into seven clusters, the foundation everything else is tailored to. Layer two is the five-stage journey that carries those personas from awareness to autonomy. Layer three is labs: the pair programming, prompt dojos, and showcases that turn framing into practice. Layer four is measurement: the KPIs, the DevEx signal, and the Center of Excellence that sustains the cadence after the champions step back. Skip the foundation and adoption falls back to the 23% baseline that programs without structure report.

> DEFINITION
>
> Five stages, one exit criterion each: preparation closes a nine-item gate, awareness frames augmentation over automation, quick wins land inside 72 hours, fluency reaches 60% weekly use, and autonomy retires the program into a community of practice. The stack underneath, personas, stages, labs, measurement, only holds if each layer rests on the one below it.

## References

Primary sources for the stage model, the change-management basis, the augmentation-framing evidence, and the GitHub Copilot platform mechanics referenced in this chapter.


- GitHub Copilot Documentation, agent mode, .github/copilot-instructions.md, and MCP, the platform mechanics the program builds on.
- Customize AI in Visual Studio Code, how repository instruction files and custom context shape agent behavior.
- Usage-based billing for individuals: GitHub AI Credits, the billing model the preparation gate budgets against, one credit is one US cent.
- Prosci, ADKAR change management model, the change-management basis for sequencing awareness before adoption.
- The four stages of competence, the competence progression the five stages mirror.

- BJ Fogg, Tiny Habits, the habit-window basis for the 72-hour quick-win target.
- MIT Sloan Management Review, AI in the Workplace, the augmentation-versus-automation framing evidence.

---

**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

03 LEARNING TRACKS

# Literacy tracks by cluster and Copilot tool: from autocomplete to orchestration.

Literacy is not one curriculum. The 24 SDLC roles split into seven clusters, and each cluster works with a different GitHub Copilot surface. An engineer lives in agent mode inside the editor. A platform engineer generates pipelines. A product manager builds an agent in Copilot Studio. This chapter maps each cluster to its primary Copilot tool, its first quick win, and the model and context choices that make the tool pay for itself.

## Engineering: from autocomplete to Agent Mode orchestration

Engineering is the largest cluster: six roles, backend, frontend, full stack, junior, mobile, and architect. The primary surface is GitHub Copilot Agent Mode and Chat. The literacy track moves through three levels. Level one is autocomplete: accept or reject inline suggestions. Level two is Chat: ask about the open file, request a refactor, generate a test. Level three is agent mode: hand Copilot a task and let it plan, edit across files, run the terminal, and open a pull request. Most developers stall at level one because no one showed them the next two.

The first quick win is deliberate and small. Take a low-complexity backlog item and drive it from branch to pull request without leaving the editor. That is the Stage 2 exit criterion for the cluster, and it is designed to land inside the 72-hour habit window. The depth metric that matters here is agentic actions per week: the count of multi-step agent mode sessions per developer. It is the number that separates a team that autocompletes from a team that orchestrates.

DEFINITION

The engineering track in one line: level one accepts suggestions, level two asks Chat, level three hands agent mode a task and reviews the pull request. Adoption stalls when a team never leaves level one.

## Repository context with .github/copilot-instructions.md and MCP

Agent mode is only as good as the context it can see. Two mechanisms carry that context in GitHub Copilot. The first is a repository instructions file, .github/copilot-instructions.md, committed to the repo. It tells Copilot the conventions the codebase already encodes: how exceptions are handled, how tests are organized, which libraries are approved, what the build command is. Every developer on the repo inherits the same instructions, so the agent stops guessing at facts the team already knows.

The second mechanism is the Model Context Protocol (MCP). MCP lets Copilot reach tools and data outside the editor: an issue tracker, a database schema, a design system, an internal API. Instead of pasting context into chat, the agent queries the source. In the onboarding program, MCP extensions belong to Stage 4, autonomy, because they assume a team that already runs agent mode fluently and knows which context is worth wiring in. The two mechanisms answer different questions: the instructions file says how this repository works, and MCP says what lives beyond it. The instructions file is week one; MCP is month four.

## Choosing models in the GitHub Copilot picker

GitHub Copilot is not a single model. The model picker exposes a menu that includes Claude Fable 5, Opus 4.8, Sonnet 5, and Haiku 4.5, alongside Gemini 3.1 Pro and the GPT-5.x family. The literacy skill is knowing which one to pick for the task in front of you. A quick inline completion does not need the most capable reasoning model. A multi-file refactor across an unfamiliar service does. Picking Haiku 4.5 for a rename and Opus 4.8 for an architecture change is the same discipline as picking the right

tool from a toolbox. The picker is one click, so the cost of a wrong choice is small, and the habit of choosing is what the track builds.

Model choice is now also a budget choice. Since June 1, 2026, GitHub Copilot bills on usage through GitHub AI Credits, where one credit equals one US cent. Different models in the picker draw different amounts of credits per request, so a team that reaches for the largest model on every trivial edit spends more than a team that matches model to task. This is why the depth metric and the model picker are taught together: orchestration without model discipline is expensive orchestration.

> DEFINITION
>
> Model literacy in one line: match the model to the task, not the task to the model. Haiku 4.5 for cheap edits, Opus 4.8 or Claude Fable 5 for hard reasoning, and read the AI Credits meter while you learn.

## Tracks for platform, data, product, and security

The other clusters each get a track built on their own artifacts. Platform, that is DevOps, cloud, DBA, and infrastructure, uses Copilot to generate infrastructure as code in Bicep, Terraform, and Bash, and its quick win is a GitHub Actions pipeline for a new service, generated with the project configuration as context. Quality and SRE use agent mode with GitHub Actions; the QA quick win is a full unit-test suite for an uncovered class, which the article clocks at about 15 minutes against 3 to 4 hours by hand.

Data and AI, meaning data engineer, data scientist, and BI analyst, work in Azure AI Foundry and Microsoft Fabric; the quick win is an exploratory analysis notebook produced in about 15 minutes. Product and design, that is PM, UX designer, and business analyst, build agents in Copilot Studio; the PM quick win is an agent that answers backlog questions from a SharePoint sheet. Security and GRC pair GitHub Advanced Security with Purview; the quick win is triaging the top three CVE findings with GHAS Security Autofix. Leadership uses Microsoft 365 Copilot for executive summaries. Same program, seven vocabularies, one set of dashboards. The

vocabulary changes by cluster, but the depth metric and the model discipline from the engineering track carry across all seven.

## References

Primary sources for the Copilot surfaces, the customization files, the model picker, and the billing model behind these tracks.


- GitHub Copilot Documentation, the reference for agent mode, Chat, and the surfaces each cluster learns.
- Customize AI in Visual Studio Code, how to write .github/copilot-instructions.md and customize Copilot in the editor.
- Usage-based billing for individuals: GitHub AI Credits, how GitHub AI Credits work and why model choice is a cost decision.
- GitHub Copilot is moving to usage-based billing, the June 1, 2026 move to usage-based billing behind the AI Credits meter.
- DORA 2025 State of AI-assisted Software Development, field evidence on AI-assisted software development practices.


---


**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes



v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

04 PART III, WINS AND METRICS

# Quick wins in seventy-two hours, measured every fortnight.

Trust is won or lost in the first seventy-two hours. Budget is kept or cut at the quarterly review. This chapter connects the two. It gives one concrete quick win per persona cluster, turns each win into a reusable pattern card, defines six KPIs that run from reach to risk, and governs the bill with GitHub AI Credits. The measurement cadence is every fortnight, because a program without numbers gets defunded.

## One concrete quick win per cluster

Habits form fast or not at all. The BJ Fogg habit window closes around seventy-two hours, so the program designs one concrete quick win per cluster to land inside it. Engineering takes a low-complexity backlog item from branch to pull request in agent mode without leaving the editor. Quality generates a full unit-test suite for an uncovered class, typically fifteen minutes of work against three to four hours by hand. Platform generates a GitHub Actions pipeline for a new service, using the project configuration as context. Product builds a Copilot Studio agent that answers backlog questions from a SharePoint sheet. Data generates an exploratory analysis notebook in fifteen minutes. Security triages the top three CVE findings with GitHub Advanced Security autofix. Leadership produces an executive retrospective summary from Teams or OneNote with Microsoft 365 Copilot.

One rule holds every win together: use the persona's own backlog, never a generic tutorial. Each quick win ships a real artifact, a pull request, a test suite, a deployment, or a working agent. That is the point of the seventy-two hour window. Past it, the literacy curve breaks and the persona quietly drops out. A win the person can show a teammate is worth more than a demo they watched.

# Pattern cards and a shared prompt library

Every quick win is captured as a one-page pattern card. The card is the seed of the team's prompt and recipe library. It records six things: the task, the model chosen from the Copilot picker, the prompt that worked, the context files it read, the artifact it produced, and the time it saved. A card is small enough to write in five minutes and specific enough for the next person to repeat the result.

The library lives in the repository, not in a wiki nobody opens. Shared conventions go in .github/copilot-instructions.md, so every Copilot session inherits them without a copy and paste. Reusable recipes become prompt files under version control. Internal systems reach agent mode through Model Context Protocol servers, so the agent queries real context instead of guessing at it.

### From a card to a convention

Cards accumulate, and the reliable ones earn promotion. When a pattern proves itself across several people, move it into copilot-instructions.md or a prompt file. Agent mode then reads it on every task, and the quick win becomes the default way the team works.

# Six KPIs from reach to risk

A program without measurement gets defunded, so six KPIs cover the full adoption signal, read every fortnight. Reach is the active user rate, the percent of licensed users active at least five days a week, with a target of eighty percent by day ninety. Depth is agentic actions per week, a count of multi-step agent mode sessions per developer, which separates autocomplete from real orchestration. Quality is the acceptance and revert rate, the suggestion acceptance rate paired with the rollback rate inside seven days, which exposes over-reliance before it becomes a liability.

Throughput is the DORA quartet: lead time, deployment frequency, change failure rate, and mean time to restore. These are lagging signals, but they are the ones that justify the budget. DevEx is a SPACE-aligned quarterly survey that includes a psychological safety question for skeptics and junior engineers. Risk is GitHub Advanced Security findings auto-fixed and merged per week, plus Defender for Cloud recommendations actioned.

### Leading signals first, lagging signals for the CFO

Read reach and depth weekly to catch a stall while it is still cheap to fix. The DORA quartet moves slowly, but it is what survives a budget review. The downside is documented: without structured onboarding, active adoption falls to twelve percent within sixty days.

## Cost governance with GitHub AI Credits

Since June 1, 2026, GitHub Copilot bills on usage through GitHub AI Credits. One credit is one US cent, so a thousand credits is ten dollars. Each request draws credits according to the model and the mode. The Copilot picker exposes the choice directly: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro, and the GPT-5.x family. Routing is now a cost decision, not only a quality one.

Governance means budgets and visibility, not a freeze. Set per-seat and per-team credit budgets, and read the usage dashboard next to the six KPIs. Route the mechanical quick wins to a fast, cheap model, and reserve the premium models for architecture and hard debugging. A quick win that burns premium credits on a trivial edit is a lesson for the pattern card, not a result to celebrate. Because each card already records the model and the credits, the library doubles as a routing guide that sharpens every fortnight.

> DEFINITION
>
> Cost governance in one rule: match the model to the task. One GitHub AI Credit is one US cent. A one-line test fix does not need Opus 4.8, and a cross-service refactor should not run on Haiku 4.5. The pattern card records the model, the credits spent, and the time saved, so routing gets cheaper and better every fortnight.

## References

Primary sources for the billing model, the customization surface, and the throughput metrics.

- GitHub Copilot Documentation, the reference platform for agent mode, the model picker, and the onboarding patterns in this chapter.
- GitHub Copilot is moving to usage-based billing, the shift to consumption billing that took effect on June 1, 2026.
- Usage-based billing for individuals: GitHub AI Credits, how credits are metered per request and per model, the basis for cost governance.
- Customize AI in Visual Studio Code, copilot-instructions.md and prompt files, the surface for the shared prompt library.
- DORA 2025 State of AI-assisted Software Development, the throughput quartet and the wider adoption evidence behind the KPIs.

---

**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

05 SUSTAIN ADOPTION

# Sustain adoption with change management and a Center of Excellence.

A rollout that ships licenses and a welcome email peaks around week two and collapses to roughly 12% active use by day sixty. Adoption survives when the program treats resistance as a signal to redirect, not an obstacle to remove, and when a durable structure owns the cadence after the formal program ends. This chapter maps the resistance, converts it in four steps, applies Kotter to the sequence, and stands up a Center of Excellence.

## The resistance map and conversion in four steps

Resistance to a new tool is not irrational. It is the response of professionals who learned to be competent one way and now face the uncertainty of learning another. William Bridges, in Managing Transitions, describes the change curve people move through at different speeds: denial, resistance, exploration, and commitment. The program has to be ready to meet each persona at any point on that curve. Each persona carries a characteristic resistance, a root fear, and a specific conversion trigger. The senior developer says agent output will lower code quality; the fear is identity as a technical expert; the trigger is showing that AI review is stricter, not looser. The product manager says this is a developer tool; the trigger is a Copilot Studio demo that solves a real product problem. The CISO says the company does not control what leaves it; the trigger is a security architecture session that shows where data stays and that GitHub Copilot Enterprise does not train models on the company code.

Conversion follows a four-step pattern drawn from Cialdini on influence and the Prosci ADKAR framework. First, acknowledge the objection without dismissing it: name it as exactly the failure the program is built to prevent. Second, offer specific evidence, not a generic claim: formally trained developers produce measurably

higher code quality than self-taught peers, so cite the number for that persona own concern. Third, give the persona control: let them define the success criterion and the review date, because autonomy reduces resistance more than persuasion does. Fourth, create the first positive experience with the resistant person as the protagonist, solving a real problem of their own live, so the win belongs to them and not to the tool.

> DEFINITION
>
> Four steps to convert resistance: acknowledge the objection without dismissing it, offer specific evidence rather than a generic claim, give the persona control over the success criterion, and stage the first positive experience with the resistant person as the protagonist. The win has to belong to the person, not to the tool.

## Kotter applied to agentic adoption

Kotter eight-step model for leading change is the most documented change framework in corporate settings, and it maps cleanly onto an agentic adoption program. Create urgency by presenting market evidence that teams which do not adopt fall behind; the executive sponsor owns this step. Form the guiding coalition by naming a Champion for each cluster and a C-level sponsor before the program starts; the CoE coordinator owns it. Create the vision by defining, concretely and per cluster, what the work looks like in six months with agents. Communicate the vision through showcases, a Teams channel, live demos, and testimonials from people already using the tool.

The second half of the model is where programs stall. Remove obstacles by clearing license, access, and security blocks before they turn into excuses. Generate short-term wins with the quick wins in weeks three to six, publicized widely. Consolidate gains by using those wins to justify expansion, new use cases, and more resources. Anchor the change in culture by putting AI adoption KPIs into team OKRs and performance criteria. The most common failure is jumping from step one straight to step five, removing technical obstacles before urgency, coalition, and vision exist. The result is a technically flawless program that nobody uses.

## The Agents Center of Excellence and community of practice

The Center of Excellence is the structure that sustains the program after the first six months. It is not a new department. It is a network of roles distributed across existing teams, modeled on the Gartner Center of Excellence framework. The CoE coordinator owns the agenda, metrics, and communication, about half an FTE for an organization of 100 to 500 developers. A Champion per cluster is a technical specialist who takes advanced training, answers the team questions, and documents use cases, roughly one Champion for every twenty people. An executive sponsor signs policy decisions and authorizes budget, without which the CoE becomes a committee with no power. A governance guardian, usually from security or compliance, confirms that new agent use cases follow approved policy before they reach production.

The community of practice complements the formal CoE. Following Wenger, McDermott, and Snyder, an effective community needs a defined domain, a body of members, and shared practice. In operation that is a dedicated Microsoft Teams channel, a biweekly 45-minute show-and-tell with two cases and open discussion, and an internal repository of prompt templates, troubleshooting guides, and documented cases. The durable convention layer lives in .github/copilot-instructions.md, where each team encodes how its codebase actually works so agent mode and MCP tools inherit it. Knowledge governance keeps the library current: every production use case is documented with the problem, the persona, the tool, the prompt or configuration, the model chosen from the GitHub Copilot picker, and the measured result, reviewed quarterly.

## The six-month roadmap and Monday-morning actions

The program runs pre-launch plus four stages across twenty-four weeks, and each transition carries a go/no-go milestone. Champions trained reaches 100% of clusters before launch, or launch waits. Licenses provisioned reaches 100% of participants on day zero, or it escalates to IT and the sponsor. Quick wins clear 70% of participants by week six, or the CoE runs extra support sessions per cluster. Active weekly use passes 60% by month three, or the team revisits the relevance of each

cluster use cases. ROI is documented with at least three measurable cases by month six, or the metrics get restructured toward more visible cases. Consumption through agent mode is billed as GitHub AI Credits, where one credit equals one US cent, so the ROI case reconciles measured productivity against a real credit line.

Three actions start the program on Monday. First, publish the AI usage policy in writing; nothing moves until it exists, and its absence is the block a CISO reaches for. Second, name two Champions per team and book their two-week advanced training. Third, schedule the first cluster-specific demo using a real artifact from the current backlog, because generic tutorials fail in week one. Onboarding is not an event. Run it like a platform, with a resistance map, a conversion pattern, a change sequence, and a Center of Excellence that owns the cadence long after the formal program retires.

> DEFINITION
>
> Monday morning, three actions: publish the AI usage policy in writing, name two Champions per team and book their two-week advanced training, and schedule the first cluster demo using a real artifact from the current backlog. Onboarding is not an event. Run it like a platform.

# References

Primary sources for the change-management models, the community-of-practice structure, and the GitHub Copilot platform this chapter builds on.


- Kotter, Leading Change, the eight-step model applied to the adoption sequence.
- Cialdini, Influence: The Psychology of Persuasion, the basis for the four-step conversion pattern.
- Prosci ADKAR, the change-management framework behind persona conversion.
- William Bridges, Managing Transitions, the change curve the resistance map follows.
- Wenger, McDermott and Snyder, Cultivating Communities of Practice, the model for the community of practice.

- Gartner, Infrastructure and Operations, the Center of Excellence framework.
- GitHub Copilot Documentation, the reference platform for agent mode, copilot-instructions.md, and MCP.

---

**Paula Silva**

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps