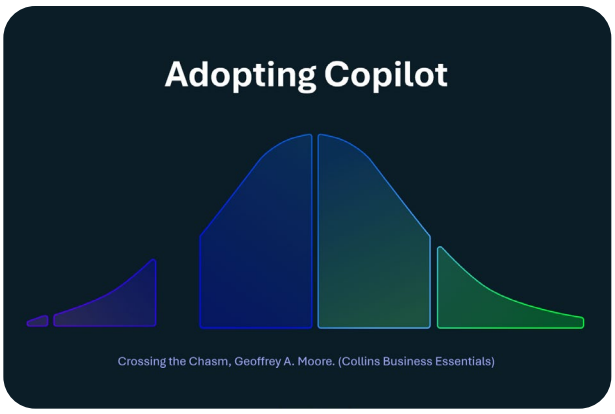
Training and onboarding developers on GitHub Copilot

WRITTEN BY GITHUB WITH ♥

How to adapt your training for generative AI tool adoption

GitHub Copilot can feel like magic, but at its core it's simply a tool—albeit a powerful and versatile one. Unlike many tools designed for specific tasks, languages, or syntax, Copilot seamlessly integrates into a developer's workflow, spanning various tasks, languages, and syntaxes. Developers work alongside Copilot to write code, generate tests, fix bugs, create documentation, and much more. To fully realize Copilot's potential, entire teams, not just individual developers, must adopt new skills. While Copilot may be a tool like any other, generative AI presents unique adoption challenges that require specific solutions.

As such, effective training and knowledge transfer are key to a successful rollout. This requires engaging the right people—those who embody the traits of innovators and early adopters, as defined by the traditional "adoption curve". They should not only be excited about the technology's potential, having used it themselves, but also motivated to teach others how Copilot enables new or augmented workflows in their daily flow.



To sufficiently bridge the gap between those early adopters and the large swath of more pragmatic, hesitant users, you'll need to create structured, repeatable onboarding processes and provide clear, tangible examples of how Copilot can solve real-world problems.

In this white paper, we will discuss:

- How Copilot fundamentally differs from other products
- The four pillars for a successful rollout
- Suggested methods for educating and enabling developers to successfully adopt Copilot
- A sample 90-day onboarding plan
- Resources for training developers to use Copilot

This white paper assumes you've already completed a pilot program and are looking to roll out Copilot to the organization at large. While we will talk about the various roles people will play, managing the rollout, and measuring success, the primary focus of this resource will be on training. For more details on running a pilot program and other technical aspects of deploying Copilot, refer to the Copilot Learning Pathway[1] and Copilot documentation[2].

1: Essentials of GitHub Copilot
2: GitHub Copilot documentation

What makes Copilot (and generative AI) unique

Generative AI can be a challenge for developers to work with, as it's unlike most tools they've used. Developers are used to deterministic environments, where a specific input always produces the same specific output. Generative AI, however, is probabilistic, meaning a specific input can result in different outputs. This is a fundamental shift, and getting the most out of Copilot requires not only an updated approach to coding, but also an understanding of how best to work with a probabilistic system.

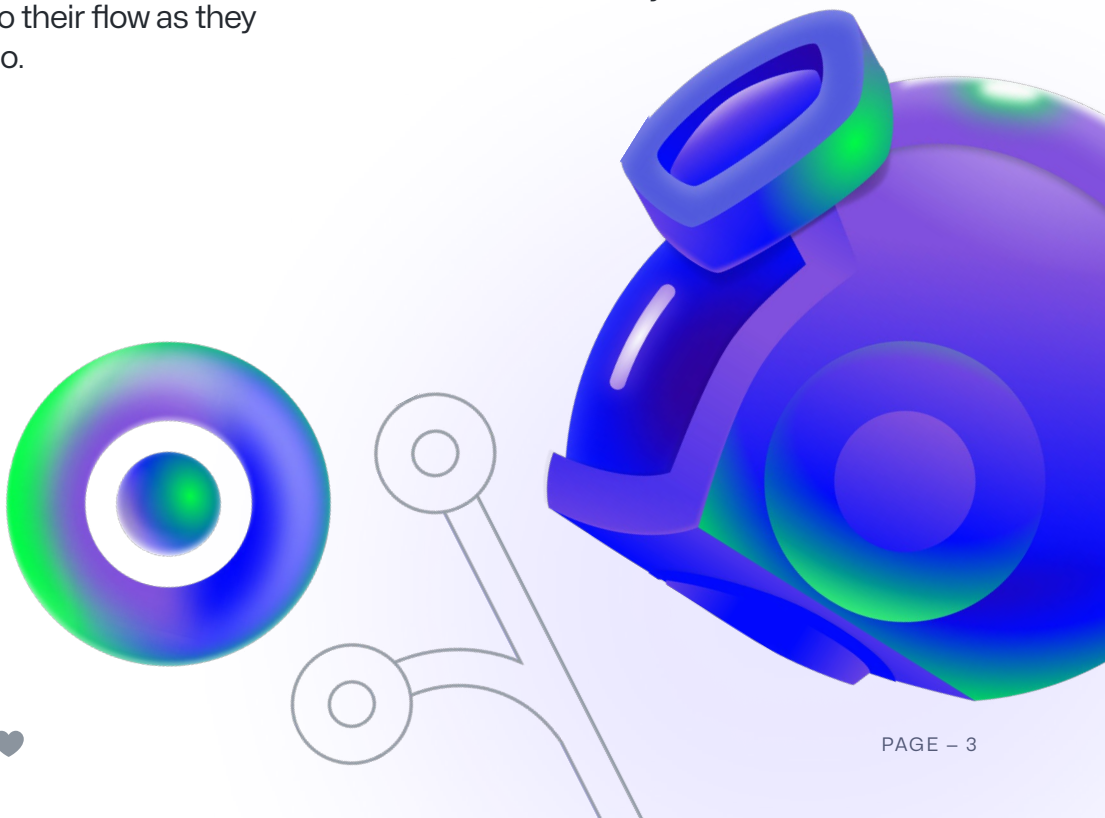
Unlike other tools, GitHub Copilot will also be pervasive throughout their development process. A new automation tool, for example, has a specific syntax and set of use cases, while Copilot stretches across languages and workloads. Copilot's use cases are varied and developers will discover new practices and ways to incorporate Copilot into their flow as they keep working with it, too.

Four pillars of successful product rollouts

There are four pillars to successfully rolling out any new tool or product to an organization:

1. **Clearly defined goals**: How can you claim success if it's not clearly defined?
2. **Champions:** Excited early adopters can influence users (developers in our case) to successfully use and adopt the tool.
3. **Reminders:** Periodic pings can help remind developers to begin (or continue) using the tool.
4. **Knowledge transfer**: Guide developers to resources on how best to use the tool.

Let's take a closer look at the first three, before really diving into knowledge transfer, which can take many forms.



Clearly defined goals

It might seem obvious, but in order for a project to succeed, success needs to be clearly defined. When rolling out Copilot, you'll want to start by defining the changes and improvements you're targeting in developer experience (DevEx), productivity, time to market, and other criteria.

A good place to start is the SPACE framework[3], an industry standard for gathering metrics.

Champions

Champions can be anyone in your organization viewed as influential, from manager to individual contributor. They are key players who help generate excitement and onboard developers to using new tools and techniques, and they can be found throughout the organization.

While there may be some overlap, champions and executive sponsors play different roles. Executive sponsors are typically higher-level leadership who drive adoption and decide the direction of the organization, while champions interact with developers on a daily basis, and are who developers look to for guidance and support.

At the same time, there are specific roles you will want to focus on. For example, a developer's response to a new tool is often reflective of the engineering manager's opinion. As such, it's vital to ensure that engineering managers have positive feedback and expectations of Copilot, because they can be some of your most effective champions.

Every team also has senior level developers who others look up to and emulate. These developers define best practices, identify patterns, and drive how development takes place on their team and often hold a large amount of sway over teams.

Champions should be visible and involved throughout the entire rollout process and beyond. Champions need to have the appropriate level of knowledge, skills, and experience to advocate for other developers. Champions can validate their skills via the Copilot exam[4] providing a clear way to differentiate champions from users. Champions can contribute in a variety of ways from leading peer-to-peer learning sessions to defining and documenting use cases and best practices.

Reminders

After a developer gains access to Copilot, they'll receive an email with instructions on how to install the extension into their preferred IDE. Most developers are extremely busy, though, and the most common reaction to that initial ping is to move on to the next task at hand. This is where reminders come into play.

Periodically remind your developers to install and begin using the tool. Reminders can be as basic as periodic messages in Slack channels or emails, and also provide an opportunity to pass along additional information. While it might feel like you're just bothering them, research shows that reminders increase adoption[5]. They also serve as great opportunities to share resources, videos, and documentation to help developers have a good first experience.

3: Measuring enterprise developer productivity

4: GitHub Certifications

5: MIT GenAI, "Generative AI and the Future of Human Creativity," 2023

Knowledge transfer: a multi-modal, long-lasting strategy

Copilot will be pervasive in your developers' workflows, so your training strategy cannot be reduced to a single event or modality. As your developers' experience with Copilot grows, they will discover new workloads and approaches they can use to get the most out of the tool. You'll want to share these learnings with others, so make your training both multimodal and long-lasting.

Let's look at how both formal and ad-hoc training will help your developers successfully adopt Copilot.

IMPORTANT: While "Don't Repeat Yourself" (DRY) is a common methodology for writing code, the exact opposite is true for training. Different learners will connect with different styles at different times throughout the lifecycle. You'll want to share information in multiple locations and modes to meet users where they're at.

Formal training

Formal training is guided or shaped by experts and has clearly defined learning objectives and measurable outcomes. With these goals in mind, formal training can promote consistency and equity across organizations.

When you build your formal training plan, you'll want to have an outline of the desired outcome, the knowledge to be transferred, and the skills your developers will need to acquire. Take into account the languages and frameworks they're working with, as lessons based on their day-to-day environment will be more impactful. Code used in training doesn't need to be production code or overly complex, but it should be representative of their daily work, and can require some team-by-team customization.

Let's look at a few types of formal training that can help.

Workshops:

Workshops provide hands-on guided instruction by someone experienced with Copilot and are a great way to introduce developers to Copilot. An expert leads learners through a series of tasks, exploring some of the common tasks Copilot can assist with and best practices for getting the most out of the tool. Workshops help shape the skills developers need to learn and gain confidence in their abilities to best take advantage of the tools available to them. Learners also have the opportunity to ask questions, and explore in an environment where they are set up for success.

Hackathons:

To host a hackathon, give developers a clearly defined task with limited guidance on how to achieve it and high-level instructions on how to use GitHub Copilot. After that, it is up to them to determine how best to resolve the issue. Many developers prefer this type of approach as they tend to want to "get their hands dirty." Make sure to have mentors available to answer questions

and point attendees in the right direction.

When planning a hackathon, ensure the challenge presented is achievable in the allotted time and has clearly defined goals. Presenting a scenario at an afternoon hackathon that would take multiple days to complete only leads to frustration. Similarly, you need clearly defined goals to help keep developers focused on a path.

Backlog items like updating documentation, handling library version upgrades, and ensuring accessibility can be good candidates for hackathon projects. We also have a packaged hackathon[6] organizations can use which provides a series of challenges where learners explore Copilot while building an application.

Asynchronous or on-demand training:

Build out a library of on-demand, curated resources to support developers with specific questions or looking for a jumpstart. These resources can come from a variety of sources including GitHub Docs, the GitHub YouTube channel, the GitHub Copilot Trust Center, and many more, which are continually updated with new content. To start, see the curated list of resources included in the appendix.

Organize your list either by topic or by a learner's journey (moving from basics, through intermediate content, and on to advanced). Make it easily accessible to developers from within their workflow, that way it's central to the onboarding process of new developers as well as ongoing development. Review resources regularly and update them as new features roll out or organizational changes are made. Designate a core group of individuals to keep this library up to date while also allowing others to recommend artifacts from other sources.

Ad-hoc and informal knowledge transfer:

While formal knowledge transfer covers the information common to all organizations, your ad-hoc and informal knowledge transfer covers your organization's unique libraries, challenges, and approaches to creating software. The workloads where Copilot shines for you, and the approach necessary to generate those quality responses, will be particular to your company, and even to individual teams. You want to have a strategy for where and how developers can share these lessons with one another.

Encourage your teams and developers to employ various knowledge sharing methods to share learnings with one another at all times. For example, if a developer discovers that Copilot generates higher quality suggestions when they have the schema file for a database open as they create a query, they don't need to wait for the next team call to share their experience.

We suggest some modes of ad-hoc knowledge transfer below, but they are illustrative, not exhaustive. Developers could share their experiences during standups, team meetings, or any other medium where they engage with one another.

Wikis and discussion forums:

Nearly every project and organization has an informal platform for developers to share and document their experiences with the codebase. Encourage developers to add their learnings and ask questions about GitHub Copilot in these locations. You can gamify sharing, especially as excitement naturally wanes after the first few months, to help keep your docs fresh and continuing to grow and evolve. You could also incentivize contribution by recognizing groups, celebrating contributions, and even providing swag to key contributors.

6: GitHub Copilot Hackathon Samples

Roll out your knowledge transfer plan

Your organization's approach to rolling out your knowledge transfer plan should meet developers where they're at, across various media and modes, and at various points during their journey. There isn't a "one-size-fits-all" approach, as different teams and even individual developers will have varying responses and connection points. Don't limit your training options to a single platform or modality, but ensure your approach is pervasive and long lasting.

Contact your sales representative to engage with GitHub about potential opportunities to train and/or certify your developers, or host hackathons and workshops.

Recommended training plan to support adoption

Different types of training will be needed at various stages, but the exact timing and approach may vary between organizations. The suggested calendar below should provide insights as to how best to approach this process.

Onboarding
(T-45 days to rollout)

Rolling out any product starts before access is enabled for users. This includes generating awareness and excitement about the product, and the initial training. Because your developers will likely have limited experience with Copilot, training will typically be more formal and higher level.

- **T-45 days:** Determine how to measure success and by what methods success will be measured. This could be anything from productivity increases, improved code acceptance rates, and developer satisfaction. For more information, review this GitHub Blog article on quantifying GitHub Copilot's impact on developer productivity and happiness[7].
- **T-30 days:** Identify and engage champions. Ensure they're onboard and have received the proper training to support their teams.
- **T-14 days:** Send announcements and a selection of asynchronous content such as videos and blog posts. The goal is to allow developers to have access to resources and time to explore them in advance.
- **T-7 days:** Host a hands-on, instructor-led workshop to answer user questions.

Adoption
(rollout to 90 days)

During the adoption period—the first 90 days—developers will begin regularly using the product in real-world scenarios, where they'll need access to information quickly. As they gain experience, they'll start to have more specific questions, which you'll want to plan for.

- **Launch day:** Ensure the following are created and accessible for all developers:
  - Slack channel for asking questions
  - Asynchronous resources, including blog posts, videos, and online courses
  - A wiki for developers to share learnings with their teams and the broader organization

7: Research: quantifying GitHub Copilot's impact on developer productivity and happiness

- **T+14 days:** Host a hackathon. Once developers have gained a little experience with Copilot, host a dedicated session where they can focus solely on using Copilot to solve a problem representative of their day-to-day work.
- **T+60 days:** Host a lunch and learn session where front-line developers share their learnings about using Copilot "in the real world." These types of events allow developers to learn best practices for using Copilot from one another.

Optimization and sustained efficiency (90 days and beyond)

After the initial adoption phase, developers will continue discovering new capabilities and solutions unique to their scenarios as they grow more comfortable using Copilot. At the same time, tool adoption tends to lose steam after initial launch, so make sure to continue the knowledge sharing and transfer process, especially as new features are released.

Focus on the following efforts regularly:

- Update wikis to ensure information is still relevant and accurate.
- Continue to host lunch and learns and similar events on a quarterly basis.
- During internal conferences, always ensure there are sessions focused on Copilot.
- Have a process for training your developers as new features are released.

Additional resources

- GitHub Trust Center
  Answers to all your questions about security, privacy, compliance, and transparency with GitHub Copilot
- Copilot Chat Cookbook
  Curated examples of prompts to use with GitHub Copilot Chat
- Copilot hackathon
  Ready-to-use hackathon to learn GitHub Copilot that can be run locally or in a codespace
- Modernizing COBOL with GitHub Copilot
- Why developer satisfaction is your best productivity metric
- Increasing collaborative development with AI
- Taking GitHub Copilot to the stars, not just the skies