WHITEPAPER



Agentic Development with GitHub Copilot

INTERCEPT

© November 2026

Introduction

As organisations seek to derive greater value from AI, agentic development marks a shift from passive tools to active collaborators.

Software development is reaching an inflection point. Developers can now delegate connected tasks to agents that work in the background or in parallel, while they provide direction, review the work and approve the outcome.

The opportunity extends beyond writing code faster. Agentic development changes how teams divide work across planning, implementation, testing, security and operations. It allows developers to spend more time on architecture, collaboration and decisions that require human judgement.

This evolution is particularly relevant for technology leaders, architects and engineering teams seeking to close the gap between AI experimentation, production use and measurable business impact.



Contents

01 The past, present and future of software development

02 The Current State of Agentic Development

03 Agentic Development in Practice

04 The business value of agentic development

05 Use cases

06 Preparing Your Team for Agentic Development



1. The past, present and future of software development

Traditionally, software development has centred on developers writing and reviewing code themselves. The IDE supports this work with features such as debugging, syntax highlighting and code completion, but the developer remains in control of the process.

This is still how many development teams work today.
However, the role of AI in software development is changing. What began with code suggestions inside the IDE is developing into agents that can carry out connected parts of the software development lifecycle (SDLC).

This shift from suggestion to execution is known as agentic development: a software development approach where AI agents actively carry out development tasks, rather than merely helping a developer write individual lines of code. Instead of waiting for a prompt at every step, agents can receive a high-level objective and work through the required actions with greater independence.



Example

Through the GitHub Copilot app, developers can start multiple development tasks in the back-ground on the same repository and review the results when they are ready. This is already a form of agentic development, even without specialised agent teams or orchestration.

Three broad waves illustrate the evolution from code completion to agentic development, and what may come next.

WAVE 1

Pair programmer
(2021 onwards)

GitHub Copilot began with code completion inside the IDE. It predicted the next line or block of code as the developer worked. The interaction was synchronous: the developer remained present and directed every step.

WAVE 2

Peer programmers
(present)

This represents the current wave of AI-supported software development. GitHub Copilot is no longer limited to suggesting the next piece of code. Developers can assign broader tasks to the Copilot Coding Agent, much like assigning a GitHub Issue to a teammate.

Developers can assign well-defined development tasks to the Copilot Coding Agent, which can research the repository, implement changes, run tests and prepare a pull request for review

This is agentic development: developers set the objective, context and boundaries, while agents perform more of the procedural work under human supervision.

WAVE 3

Teams and agents
(emerging)

The emerging third wave extends agentic development from individual agents to hybrid teams of humans and multiple specialised agents working across the SDLC. Their role extends beyond coding to planning, issue decomposition, testing, security, documentation, monitoring and operational troubleshooting.

Agents work with other agents, orchestrated by humans: Engineers guide these agents, set boundaries, create specifications and delegate work. As multiple agents work in the background, engineers focus more on architecture, coordination and review, but remain accountable for the outcomes and approval of pull requests. The goal is faster delivery. However, these workflows are still emerging and introduce additional cost, complexity and governance requirements.

Note

These waves describe the development of the technology, not the three stages that every organisation has already completed. Many development teams still use AI mainly for code completion and chat. Others work with supervised agents, while a smaller group is experimenting with more autonomous and multi-agent workflows.

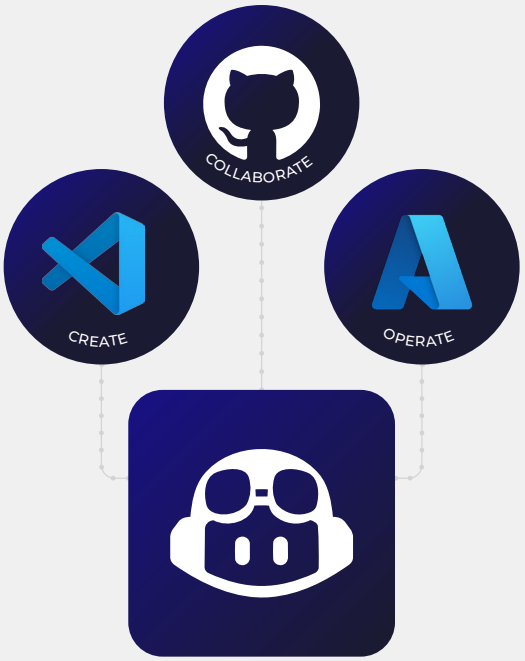
Why this shift matters

Many development bottlenecks sit around the code itself. Understanding requirements, tracing bugs, writing tests, reviewing pull requests and keeping pipelines healthy all require time and coordination.

Agentic development allows AI to contribute across these connected parts of the SDLC. GitHub Copilot can act as both a coding assistant and a workflow partner, carrying an objective through a sequence of actions.

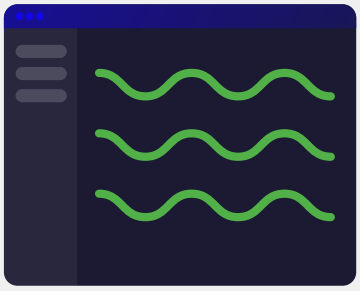
- Developers set objectives, provide context, establish boundaries and review outcomes.
- AI agents perform more of the procedural work required to turn intent into a working result.

This allows developers to spend more time on decision-making and validation, while agents handle more of the implementation.



Where is your development team today

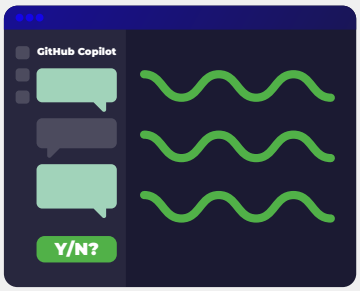
The three waves show the broader evolution of software development. In practice, adoption happens more gradually. The following eight stages provide a way to assess how your developers currently work with AI.



Stage 1

Limited AI assistance

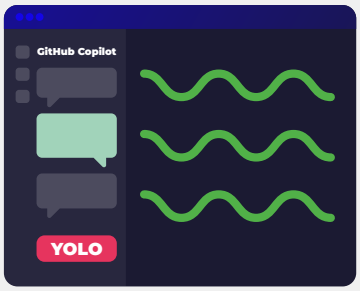
Developers use code completion or chat occasionally, but perform and connect the individual steps themselves.



Stage 2

Supervised agent in the IDE

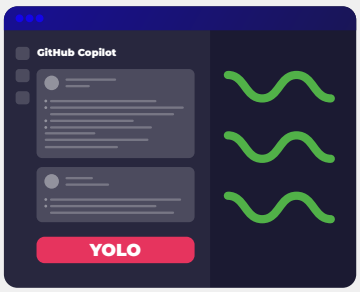
An agent can inspect files, make changes and use tools, but requests approval before taking actions.



Stage 3

Autonomous agent in the IDE

The agent performs more actions independently, while the developer monitors its work and reviews the results.



Stage 4

Agent-led development in the IDE

Developers work primarily through objectives, plans and reviews. The agent handles more of the implementation.



Stage 5

Single agent outside the IDE

A developer directs an agent through the command line and reviews the resulting changes and output.



Stage 6

Multiple parallel agents

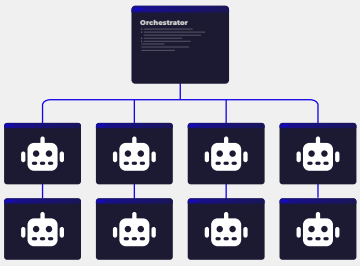
Several agents work simultaneously on separate tasks, coordinated by the developer.



Stage 7

Agent coordination at scale

Developers manage larger groups of agents. Context, dependencies and quality control become central concerns.



Stage 8

Agent orchestration

An orchestration system distributes and coordinates work across multiple agents. Agents work with other agents, while developers define the objectives, boundaries and validation requirements.

Note

Developers within the same team may work at different stages. Some may primarily use code completion and chat, while others already work with supervised or more autonomous agents. These stages are not a maturity ranking that every team must complete. The appropriate level depends on the task, codebase, risk and available controls.

2. The Current State of Agentic Development

Expectations for agentic AI are rising quickly, but implementation remains uneven. Organisations differ considerably in how far they have progressed and whether the operational and governance foundations for responsible adoption are already in place.

Expectations are ahead of implementation

Today, **23%** of business and technology leaders report at least moderate use of agentic AI.

Within two years, **74%** expect their organisations to have reached that level.[1]

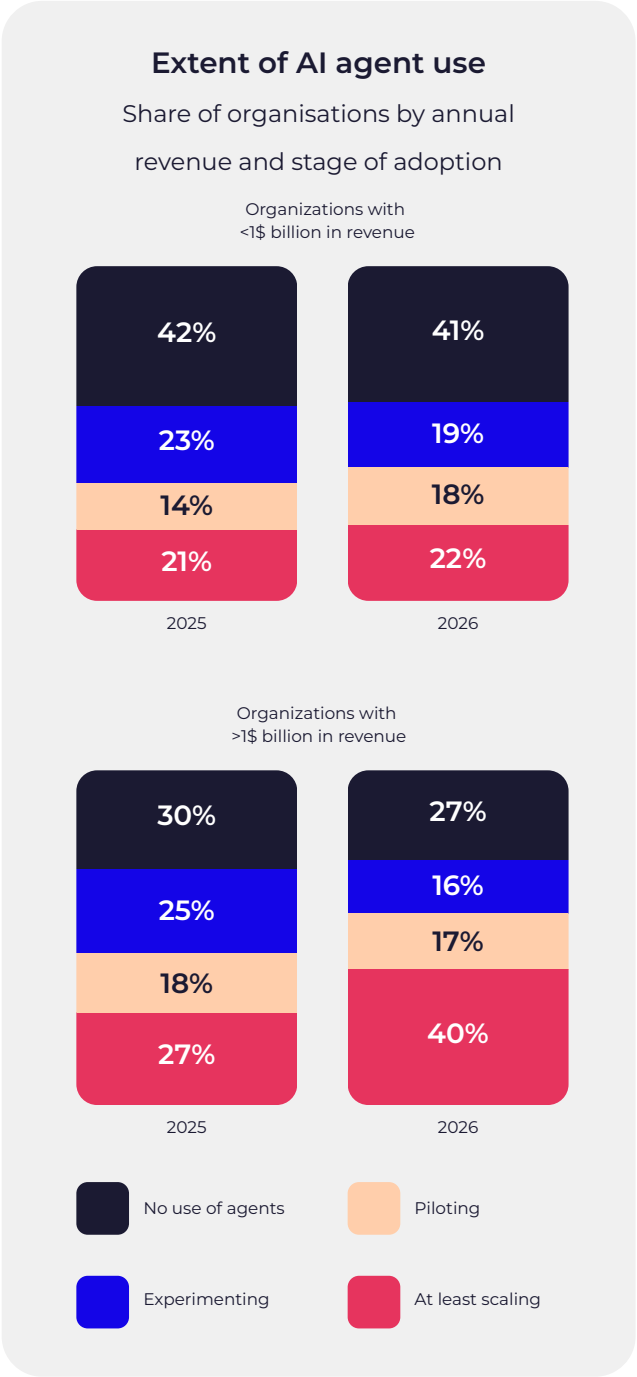
Microsoft projects that more than **1.3 billion** AI agents will be deployed globally by 2027.

Most organisations, however, are still at the beginning of this transition.

Adoption differs by annual revenue

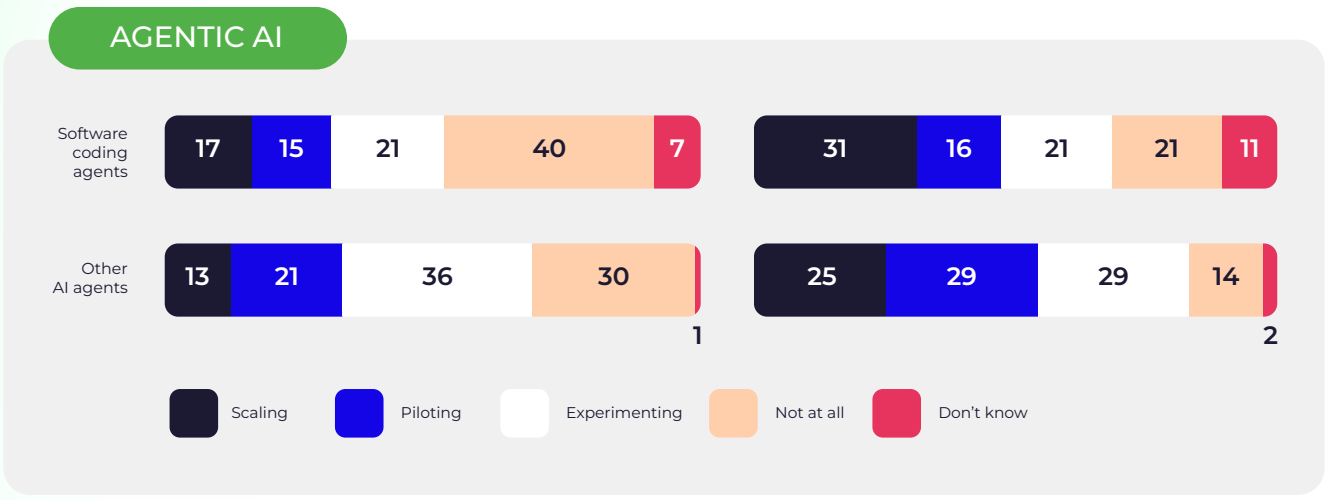
Among organisations with at least $1 billion in annual revenue

- 31% report scaling software coding agents
- compared with 17% of organisations with less than $1 billion in annual revenue.[3]





The difference is also visible at the other end of the spectrum. 40% of organisations with less than $1 billion in annual revenue report no use of coding agents, compared with 21% of organisations with annual revenue of at least $1 billion. However, 21% of both groups are still experimenting.[2]



These figures show that adoption is progressing, but remains far from consistent across the market.

GitHub adoption shows both scale and a maturity gap

GitHub Copilot grew from more than 15 million users in April 2025 to 50 million by mid-2026.[4] GitHub has become a central platform for agentic development. Major coding agents now run on the platform, and one in three pull requests involves an agent.[4]

The way developers use Copilot also matters. Across engaged Copilot Business and Enterprise users,

62.1%

still use Copilot primarily for code completions or IDE-based assistance, rather than working with GitHub-based agents or multi-agent workflows.[5]

For many organisations, adoption has therefore not yet progressed to GitHub-based agent or multi-agent workflows, leaving much of the potential value of agentic development untapped.

Measuring adoption requires organisations to look beyond whether developers use AI and assess how they use it.



Governance is not keeping pace

Only

21%

of organisations have a mature governance model for autonomous agents.[1] Such models establish boundaries, monitoring and auditability around the work agents perform.

As agents take responsibility for more connected tasks, this gap becomes increasingly important. Greater autonomy requires clearer accountability, stronger controls and defined validation processes.

The direction is clear, but implementation is still developing

Agentic development is moving into real development workflows, but adoption alone does not indicate maturity.

> Many organisations still need to move beyond basic AI assistance and establish the processes, controls and governance required for responsible agentic development. The challenge now is to turn growing usage into consistent and measurable value.

Sources

1 Deloitte, The State of AI in the Enterprise 2026
2 Microsoft, Security in the Agentic Era: A New Paradigm
3 McKinsey & Company, The State of AI: Global Survey 2026
4 Microsoft, Fiscal Year 2025 Third Quarter Earnings Conference Call and Fiscal Year 2026 Fourth Quarter Earnings Conference Call
5 GitHub, From code completions to agents: what AI adoption is really worth
6 Deloitte, AI Agents Are Only the Beginning: Deloitte Survey Shows Why Autonomous Operations May Be the Real Destination

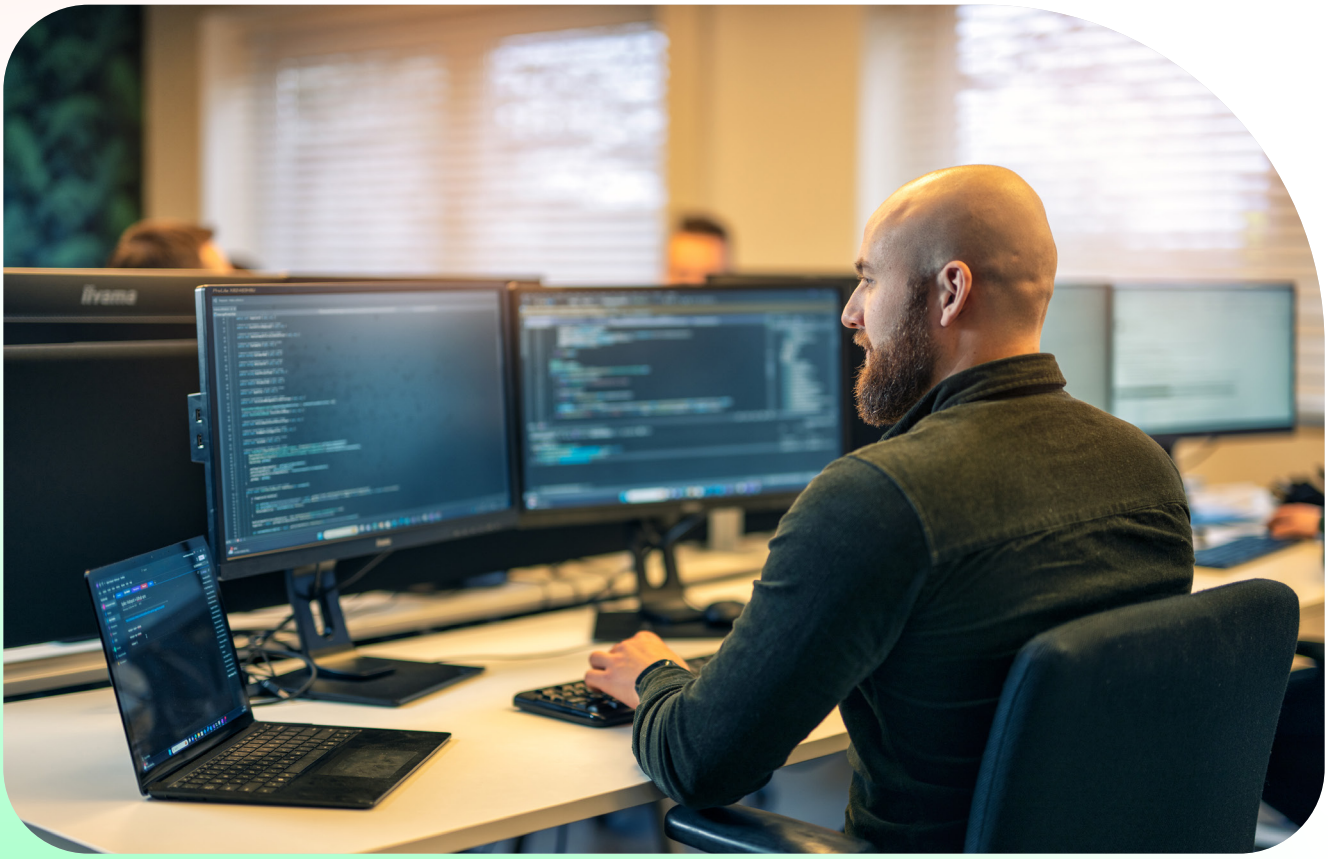
3. Agentic Development in Practice

Agentic development changes how software teams divide and coordinate work. Developers no longer need to perform every individual step themselves. They can define an objective and delegate connected development tasks to agents.

A team using agentic development still includes human roles such as product owners, engineers and QA or operations specialists. They define the work, provide context and delegate specific tasks to agents.

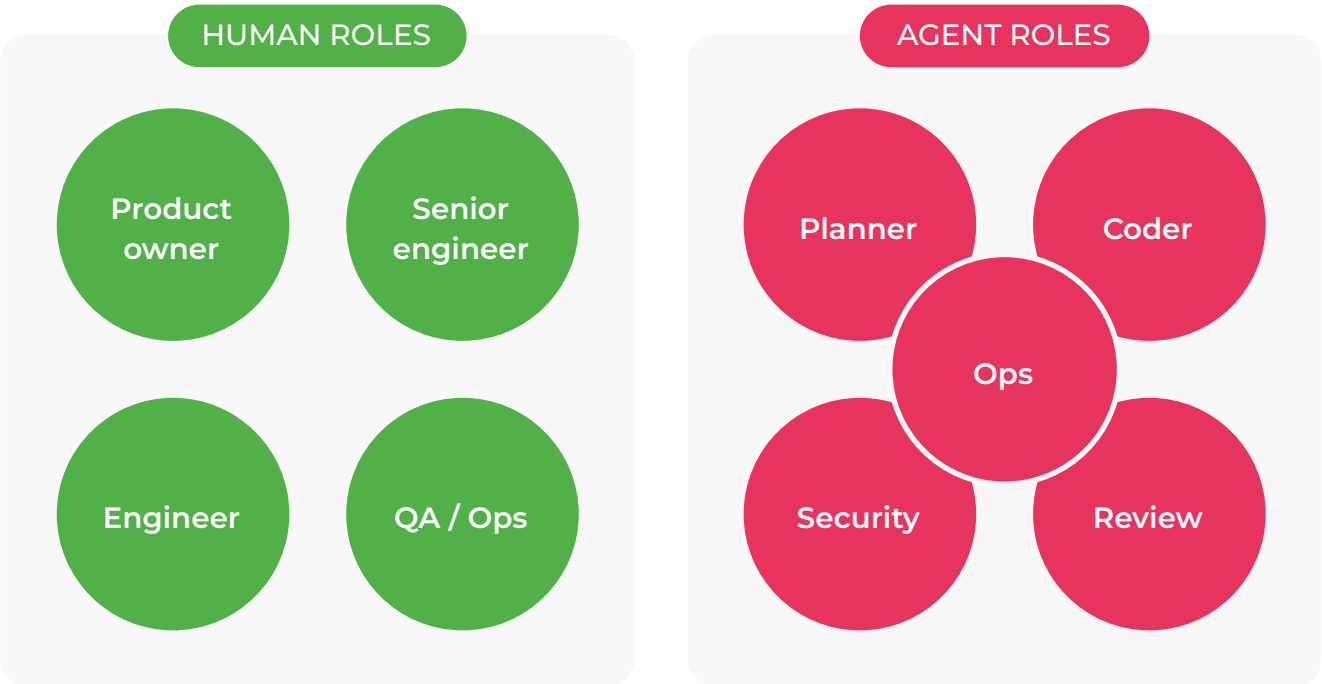
Agents can also have **specialised roles**. For example, a planner can propose an approach before a coding agent implements it. A security agent can check the changes for vulnerabilities, while a review agent assesses quality and maintainability.

This creates a workflow in which each agent contributes to a defined part of the objective, while developers coordinate the work and approve the outcome.



Human and agent responsibilities

- AI agents can gather context, plan an approach, modify code, use tools and respond to results.
- Developers define the objective, provide context and determine the boundaries within which agents can operate. They review the output, make decisions that require judgement and remain accountable for the result.





The agentic development workflow

An agentic development workflow consists of four connected phases. The agent gathers context, plans the work, implements the solution and reviews the result before it is submitted for human approval:

1	**Research**	The agent examines the objective, codebase, constraints, dependencies and potential risks.
2	**Plan**	It determines the approach, divides the work into tasks, selects the required tools and defines how the result will be validated.
3	**Implement**	The agent makes the planned changes and continuously runs the relevant tests and checks.
4	**Review**	It assesses the result for quality, security and correctness, then refines the work before human approval.

The developer remains involved when the work requires additional context or judgement and approves the final outcome.

Without context, an agent has to guess

Giving an agent a task without context is like assigning work to a new developer without explaining the codebase or how the team works.

Before an agent can complete a task reliably, it needs to understand the project structure, coding standards, architectural decisions and expected outcome.

GitHub Copilot can receive this context in several ways:

- Repository instructions describe the team's coding conventions, project structure and development practices.
- Skills provide reusable instructions for specific tasks, such as creating an API endpoint or performing a security review.
- Model Context Protocol (MCP) servers connect the agent to approved tools and information outside the immediate codebase, such as internal documentation or testing environments.

Together, these sources give the agent the context it needs to work within the team's actual development environment.

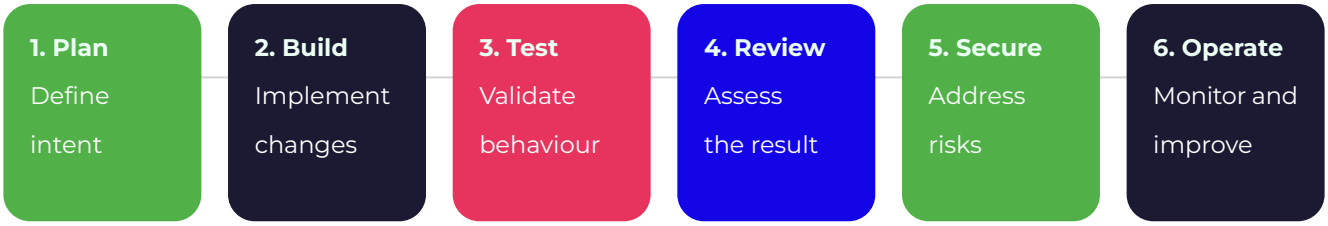
Agentic development across the SDLC

AI is turning the traditional SDLC on its head, with GitHub Copilot helping make that shift possible. Imagine a developer's typical workday. A bug report comes in. They investigate the problem, change the code, write tests, open a pull request, respond to feedback and monitor the pipeline. Many steps, many context switches.

With GitHub Copilot and its agents, the human remains at the wheel while agents carry more of the work between these steps. This creates a faster, more connected development workflow without removing human control.

GitHub Copilot Coding Agent sits at the centre of this shift. It can complete a defined task from issue to pull request, while the developer remains responsible for reviewing and approving the outcome.

Here is how this workflow can unfold across six phases of the SDLC:



Developers set direction and approve outcomes. Agents carry out defined work between human checkpoints:

Phase	Developer responsibility	Agent contribution
Plan	Define the objective, scope and acceptance criteria	Gather context and propose an implementation plan
Build	Review the approach and key design decisions	Create a branch and implement the planned changes
Test	Determine what needs to be validated	Generate tests, run checks and respond to failures
Review	Assess the changes and approve the outcome	Summarise the work and flag potential issues
Secure	Decide how identified risks should be handled	Identify vulnerabilities and propose targeted fixes
Operate	Confirm the production impact	Analyse pipeline failures and propose corrections

Across the SDLC, the balance remains the same: developers provide direction and make the final decisions, while agents carry out defined work within those boundaries.

4. The business value of agentic development



For technology leaders, the value of agentic development appears in measurable outcomes: shorter delivery cycles, stronger quality and security, fewer bottlenecks and better visibility into progress.

The depth of adoption influences the value a team can achieve. Code completion supports individual tasks. As teams move towards delegation, developers can assign complete tasks to agents, while more advanced teams can coordinate several specialised agents across the SDLC. The gains then extend from individual productivity to team-wide delivery.

Faster delivery with fewer bottlenecks

When agents handle more procedural work, developers spend more time making decisions and less time chasing details. A bug fix can be implemented and validated with tests before it reaches the main branch. A pull request can include a plain-language explanation of the changes and potential risks. This shortens the path from issue to approved change.

GitHub reports that engaged Copilot users merge **3.3 times more** pull requests and complete those merges **2.4 times faster** than passive users.[7] The exact impact varies by team, use case and level of adoption. (source: https://github.com/resources/insights/copilot-impact-dashboard)

As development becomes faster, the bottleneck can move elsewhere. Product teams may need to make decisions and prioritise work more quickly to use the additional engineering capacity effectively.

Better quality and lower release risk

Testing and security become part of the development workflow rather than separate checks at the end. This helps teams identify defects and vulnerabilities earlier, when they are easier and less costly to resolve.

Automated checks can identify routine issues before peer review, allowing human reviewers to focus on correctness, security risks and long-term maintainability. This shortens review cycles, makes better use of senior expertise and reduces the chance of costly problems reaching production.

The result is faster feedback, less rework and greater confidence in each release.

Greater visibility and predictability

It's essential to know when work will be completed, whether it is safe to release and where blockers are emerging.

Agentic workflows make these signals visible within the development process:

- A pull request includes a summary of the changes.
- A dependency alert includes a proposed fix.
- A failing pipeline includes the likely cause and a sugge-sted correction.

Information that previously required additional meetings or investigation becomes available alongside the work itself. This gives teams a clearer view of progress, risk and release readiness.



More time for decisions and collaboration

As agents take on routine implementation, testing and review tasks, developers gain more time for the parts of software development that require discussion and judgement.

They can work more closely with product managers, designers, testers and operations specialists on requirements, architecture, edge cases and user experience. Instead of spending most of their time on mechanical development tasks, they can focus on deciding what should be built and whether the outcome meets the intended need.

This strengthens collaboration across the development process. Developers increasingly guide agents and assess their output, while using their own expertise where it adds the most value.

More consistent development at scale

Agentic workflows can apply shared instructions, quality checks and approval rules across teams. This makes it easier to maintain consistent development practices as adoption grows.

Clear checkpoints keep developers in control of higher-risk decisions, while agents carry out routine work within defined boundaries. Organisations can therefore add capacity without giving up oversight or accountability.

The business value extends beyond producing more code. It comes from moving quality software through the SDLC with fewer bottlenecks, clearer accountability and more time for the decisions that require human judgement.

5. Use cases

The following scenarios show how developers can delegate connected tasks to GitHub Copilot Agents while retaining responsibility for the outcome.

SCENARIO 1

Tackling a complex bug

A product team discovers a bug that only appears when users move quickly between screens. A developer opens a GitHub Issue and assigns the Copilot Coding Agent. The agent creates a branch, adds test scaffolding and identifies where application state may be leaking.

In Visual Studio Code, the developer asks GitHub Copilot to trace the relevant state variable and propose a fix. After reviewing the changes, the developer uses Playwright through an MCP server to confirm that the problem no longer occurs. This allows the developer to focus on assessing the cause and proposed solution rather than tracing every component manually.

SCENARIO 1

Migrating an application to a different language

A team needs to move an active HTTP API from Python with FastAPI to C# with .NET Minimal APIs without changing its behaviour.

Copilot Agent Mode analyses the codebase, plans the architectural changes and rebuilds the endpoints in the new framework. It transforms the required data structures, adds Swagger and OpenAPI documentation and creates unit tests to compare both applications.

The developer reviews the architecture and confirms that the migrated application meets the original requirements. The agent handles much of the repetitive translation and validation work, allowing the developer to focus on compatibility, design and quality.



SCENARIO 3

Repairing a broken CI/CD pipeline

Shortly before a planned release, a GitHub Actions build fails because an update to a third-party dependency introduced a breaking change. Further deployments are blocked.

When the build fails, a configured workflow creates an issue containing the relevant logs and assigns it to the GitHub Copilot coding agent. The agent analyses the failure, identifies the updated dependency and prepares a change that pins it to the last stable version. It then opens a draft pull request, where GitHub Actions validates the correction before the developer reviews it.

The developer reviews the change and decides whether it should be merged. The team receives a tested correction without asking several engineers to investigate the same failure.

Across these scenarios, the division of work remains consistent. Developers define the objective, provide context and approve the outcome. Agents carry out connected research, implementation and validation work between those human checkpoints.

6. Preparing Your Team for Agentic Development

Agentic development is an organisational change as well as a developer shift. Successful adoption requires consistent access, practical experience and clear boundaries.

Start with a focused pilot

Choose one or two workflows with a clear objective and limited risk, such as investigating bugs, generating tests, updating dependencies or diagnosing failed pipelines.

Define the expected outcome before starting. This could include shorter cycle times, less review effort or faster recovery from failures. A contained pilot helps the team confirm the value, identify missing context and refine the controls before expanding adoption.

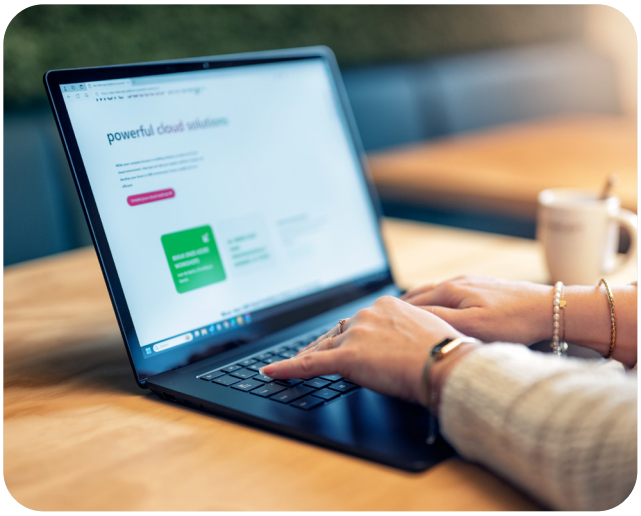
Teams also need consistent access to the required GitHub Copilot capabilities. Business and Enterprise plans provide the shared features and policy controls needed when adoption expands beyond individual experimentation.

Build practical skills and the right culture

Developers familiar with code completion may still need guidance on Agent Mode, Coding Agent, MCP servers and reviewing agent-generated changes.

Hands-on practice using the team's own codebase helps developers understand where agents perform well and where closer supervision is required. Product owners, architects, testers and operations specialists should also understand how agents affect their part of the workflow.

Leaders should position GitHub Copilot as a collaborator rather than a replacement or shortcut. Developers remain responsible for guiding agents, reviewing their work and approving the outcome.





Provide context and establish guardrails

Agents need clear objectives, acceptance criteria and access to the instructions and tools required for the task. Repository instructions, skills and approved MCP servers help agents work within the team's actual development environment.

Teams must also define which actions agents may perform independently and which require human approval. Policies should cover permissions, security, access, quality standards and acceptable use.

Clear review checkpoints, monitoring and audit trails allow teams to understand what an agent changed and intervene when necessary. Agents can propose and implement changes, but teams remain accountable for what enters production.

Measure, learn and scale

Usage alone does not show whether adoption is creating value. Track outcomes such as cycle time, time to merge, review effort, escaped defects, pipeline recovery and developer satisfaction.

Use these results to identify where agentic workflows create value and where teams need additional support. Expand gradually across teams and use cases, applying the standards and lessons established during the pilot.



Ready for agentic development with GitHub Copilot?

Many organisations already use GitHub Copilot, but not every developer or team gets the same value from it.

During an interactive CIE session, developers work through practical scenarios and learn how to apply agentic development with GitHub Copilot more effectively in their daily workflows.

REQUEST A CIE SESSION

Organised by Microsoft, GitHub and Intercept.