WHITEPAPER



# Agentic Development met GitHub Copilot

INTERCEPT

© November 2026

# Introductie

Terwijl organisaties meer waarde uit AI willen halen, verandert agentic development de rol van AI: van een passieve tool naar een actieve samenwerkingspartner.

Softwareontwikkeling bereikt een omslagpunt. Developers kunnen nu samenhangende taken delegeren aan agents die op de achtergrond of parallel werken, terwijl zij richting geven, het werk reviewen en het resultaat goedkeuren.

De mogelijkheden gaan verder dan sneller code schrijven. Agentic development verandert hoe teams het werk verdelen over planning, implementatie, testen, security en operations. Hierdoor houden developers meer tijd over voor architectuur, samenwerking en beslissingen die menselijk inzicht vragen.

Deze ontwikkeling is vooral relevant voor technologieleiders, architecten en engineeringteams die de kloof tussen AI-experimenten, gebruik in productie en meetbare bedrijfswaarde willen verkleinen.



# Contents


01 Het verleden, heden en de toekomst van softwareontwikkeling

02 De huidige stand van agentic development

03 Agentic development in de praktijk

04 Wat agentic development oplevert

05 Praktijkvoorbeelden

06 Je team voorbereiden op agentic development




# 1. Het verleden, heden en de toekomst van softwareontwikkeling

Traditioneel schrijven en reviewen developers de code zelf. De IDE ondersteunt hen daarbij met functies zoals debugging, syntax highlighting en code completion, maar de developer houdt de controle over het proces.

**Veel developmentteams werken nog steeds op deze manier.**
De rol van AI binnen softwareontwikkeling verandert echter. Wat begon met codesuggesties in de IDE, ontwikkelt zich tot agents die samenhangende onderdelen van de software development lifecycle (SDLC) kunnen uitvoeren.

Deze stap van suggestie naar uitvoering noemen we agentic development: een aanpak waarbij AI-agents actief developmenttaken uitvoeren, in plaats van een developer alleen te helpen bij afzonderlijke coderegels. Een agent kan een doel op hoofdlijnen krijgen en de benodigde acties zelfstandiger doorlopen.

**Voorbeeld**

Via de GitHub Copilot-app kunnen developers meerdere taken voor dezelfde repository op de achtergrond starten. Zodra het werk klaar is, reviewen zij de resultaten. Dit is al een vorm van agentic development, ook zonder gespecialiseerde agentteams of orchestration.

**Drie golven van code completion naar agentic development en wat daarna kan volgen.**

GOLF 1

## Pair programmer
(2021 en verder)

GitHub Copilot begon met code completion in de IDE. Terwijl de developer werkte, voorspelde Copilot de volgende regel of het volgende codeblok. De interactie verliep synchroon: de developer bleef aanwezig en stuurde iedere stap aan.

GOLF 2

## Peer programmers
(heden)

Dit is de huidige golf van AI-ondersteunde softwareontwikkeling. GitHub Copilot stelt niet langer alleen het volgende stuk code voor. Developers kunnen bredere taken toewijzen aan de Copilot Coding Agent, ongeveer zoals zij een GitHub Issue aan een collega toewijzen.

Developers kunnen duidelijk afgebakende taken toewijzen aan de Copilot Coding Agent. Die onderzoekt de repository, implementeert wijzigingen, voert tests uit en bereidt een pull request voor op review. Developers bepalen het doel, de context en de grenzen, terwijl agents onder menselijk toezicht meer procedureel werk uitvoeren.

GOLF 3

## Teams & agents
(in opkomst)

De opkomende derde golf breidt agentic development uit van afzonderlijke agents naar hybride teams van mensen en meerdere gespecialiseerde agents die binnen de SDLC samenwerken. Hun rol gaat verder dan coding en omvat ook planning, het opdelen van issues, testen, security, documentatie, monitoring en operationele troubleshooting.

Agents werken met elkaar samen onder menselijke aansturing. Engineers begeleiden de agents, stellen grenzen, maken specificaties en delegeren werk. Terwijl meerdere agents op de achtergrond werken, richten engineers zich meer op architectuur, coördinatie en review. Zij blijven verantwoordelijk voor de resultaten en de goedkeuring van pull requests. Het doel is snellere delivery. Deze workflows zijn nog in ontwikkeling en brengen extra kosten, complexiteit en governance-eisen met zich mee.

**Let op**

Deze golven beschrijven de ontwikkeling van de technologie. Het zijn geen drie fasen die iedere organisatie al heeft doorlopen. Veel teams gebruiken AI vooral voor code completion en chat. Andere werken met agents onder toezicht, terwijl een kleinere groep experimenteert met autonomere agents en multi-agentworkflows.

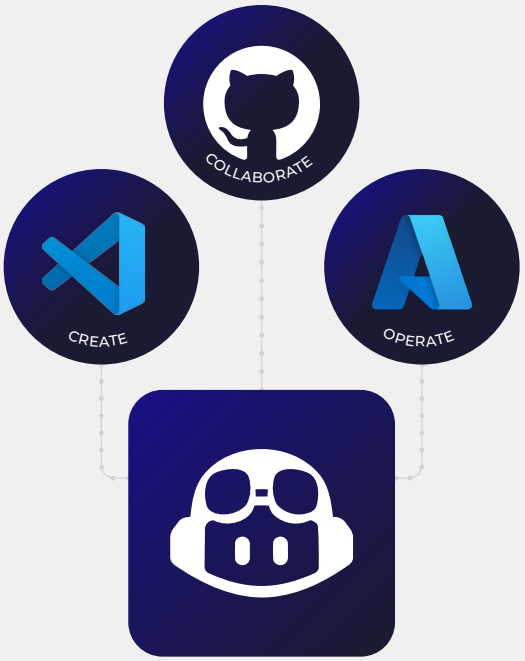
## Waarom deze verschuiving ertoe doet

Veel knelpunten bevinden zich rond de code zelf. Vereisten begrijpen, bugs traceren, tests schrijven, pull requests reviewen en pipelines gezond houden kosten allemaal tijd en vragen om afstemming.

Met agentic development kan AI bijdragen aan deze samenhangende onderdelen van de SDLC. GitHub Copilot kan zowel een coding assistant als een workflowpartner zijn en een doel door een reeks acties heen uitvoeren.

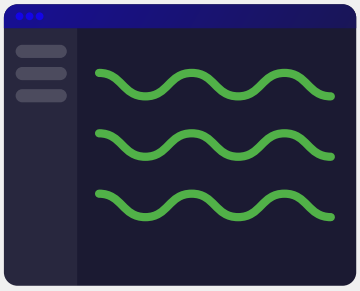
- Developers stellen doelen, bieden context, bepalen grenzen en reviewen resultaten.
- AI-agents voeren meer procedureel werk uit om een bedoeling om te zetten in een werkend resultaat.

Hierdoor houden developers meer tijd over voor beslissingen en validatie, terwijl agents een groter deel van de implementatie uitvoeren.



# Waar staat je developmentteam vandaag?

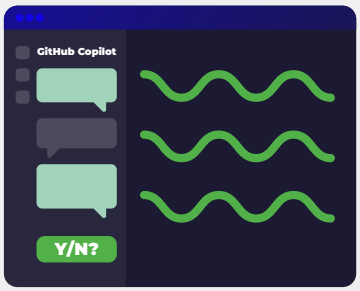
De drie golven laten de bredere ontwikkeling van softwaredevelopment zien. In de praktijk verloopt adoptie geleidelijker. De volgende acht fasen helpen bepalen hoe developers op dit moment met AI werken.



**Fase 1**

**Beperkte AI-ondersteuning**

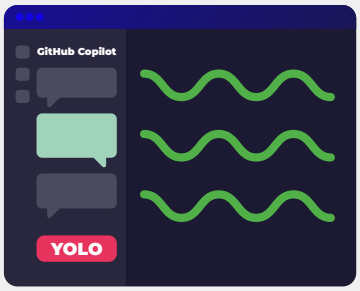
Developers gebruiken af en toe code completion of chat, maar voeren de afzonderlijke stappen zelf uit en verbinden deze handmatig.




**Fase 2**

**Agent onder toezicht in de IDE**

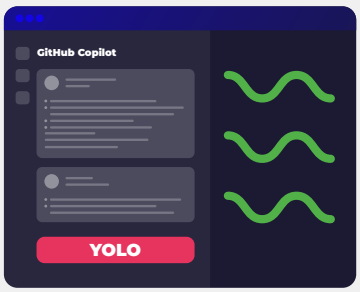
Een agent kan bestanden onderzoeken, wijzigingen uitvoeren en tools gebruiken, maar vraagt toestemming voordat acties worden uitgevoerd.




**Fase 3**

**Autonome agent in de IDE**

De agent voert meer acties zelfstandig uit, terwijl de developer het werk monitort en de resultaten reviewt.




**Fase 4**

**Agent-led development in de IDE**

Developers werken vooral met doelen, plannen en reviews. De agent neemt een groter deel van de implementatie voor zijn rekening.




## Fase 5

**Eén agent buiten de IDE**

Een developer stuurt een agent via de command line aan en reviewt de wijzigingen en output.




## Fase 6

**Meerdere parallelle agents**

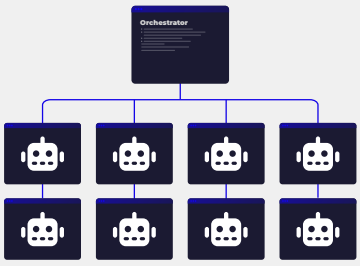
Meerdere agents werken tegelijk aan afzonderlijke taken, onder coördinatie van de developer.




## Fase 7

**Agentcoördinatie op schaal**

Developers sturen grotere groepen agents aan. Context, dependencies en kwaliteitscontrole worden centrale aandachtspunten.




## Fase 8

**Agent orchestration**

Een orchestratiesysteem verdeelt en coördineert het werk over meerdere agents. De agents werken met elkaar samen, terwijl developers de doelen, grenzen en validatievereisten bepalen.

## Let op

Developers binnen hetzelfde team kunnen zich in verschillende fasen bevinden. Zie deze fasen niet als een volwassenheidsladder die ieder team volledig moet doorlopen. Het passende niveau hangt af van de taak, de codebase, de risico's en de beschikbare controles.

# 2. De huidige stand van agentic development

De verwachtingen rond agentic AI nemen snel toe, maar de implementatie verloopt nog ongelijk. Organisaties verschillen sterk in hoe ver zij zijn en of de operationele basis en governance voor verantwoorde adoptie al aanwezig zijn.

## Verwachtingen lopen voor op de implementatie

Op dit moment meldt **23%** van de business- en technologieleiders dat hun organisatie agentic AI minstens op gemiddeld niveau gebruikt.

Binnen twee jaar verwacht **74%** dat de eigen organisatie dit niveau heeft bereikt.[1]

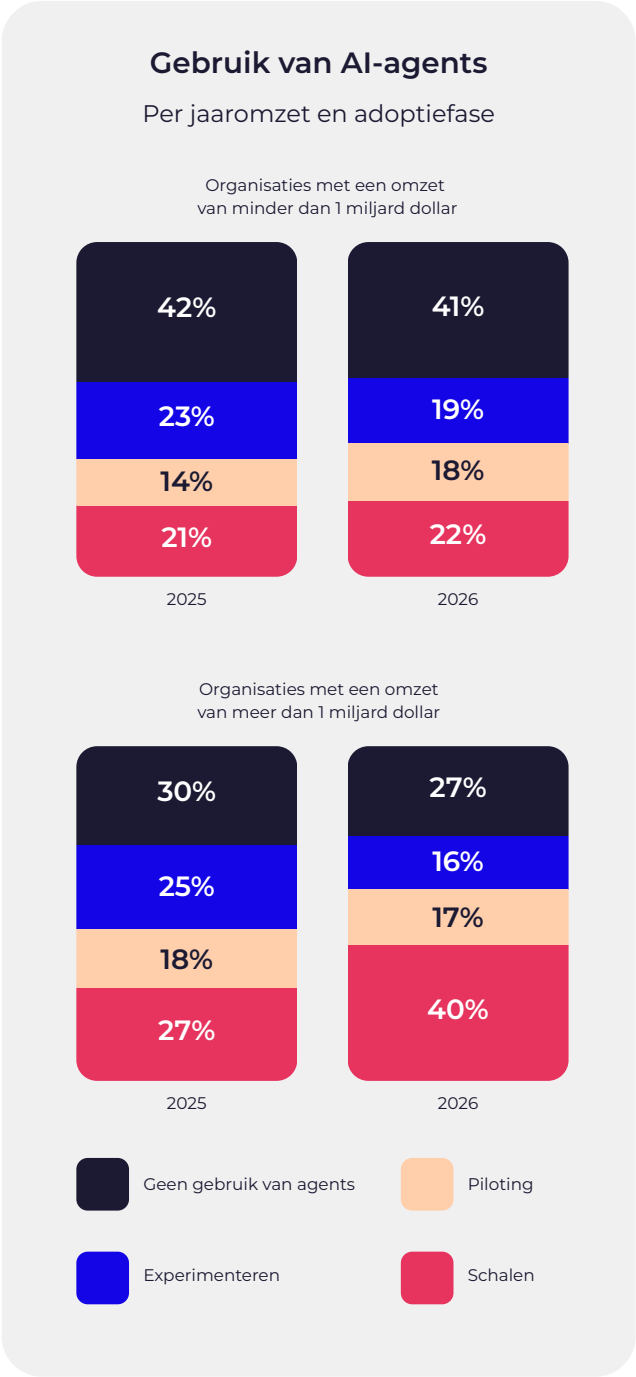
Microsoft verwacht dat er in 2027 wereldwijd meer dan **1,3 miljard** AI-agents zijn ingezet.[2]

Toch staan de meeste organisaties nog aan het begin van deze ontwikkeling.

## Adoptie verschilt naar jaaromzet

Bij organisaties met minimaal $1 miljard jaaromzet

- schaalt 31% software coding agents op
- tegenover 17% van organisaties met minder dan $1 miljard omzet.[3]






Ook aan de andere kant van het spectrum is het verschil zichtbaar. Van de organisaties met minder dan $1 miljard jaaromzet gebruikt 40% geen coding agents, tegenover 21% van de organisaties met minimaal $1 miljard jaaromzet. In beide groepen bevindt 21% zich nog in de experimenteerfase.[2]




Deze cijfers laten zien dat de adoptie groeit, maar nog sterk verschilt binnen de markt.

# GitHub-adoptie toont schaal én een volwassenheidskloof

GitHub Copilot groeide van ruim 15 miljoen gebruikers in april 2025 naar 50 miljoen medio 2026.[4] GitHub is uitgegroeid tot een centraal platform voor agentic development. Grote coding agents draaien inmiddels op het platform en bij één op de drie pull requests is een agent betrokken.[4]

Ook de manier waarop developers Copilot gebruiken is bepalend. Van de actieve Copilot Business- en Enterprise-gebruikers gebruikt

**62.1%**

Copilot nog vooral voor code completion of ondersteuning in de IDE, in plaats van GitHub-agents of multi-agentworkflows.[5]

Bij veel organisaties is de adoptie dus nog niet gevorderd tot GitHub-agents of multi-agentworkflows. Daardoor blijft een groot deel van de potentiële waarde van agentic development onbenut.

Om adoptie te meten, moeten organisaties daarom verder kijken dan de vraag óf developers AI gebruiken. Het gaat er ook om hóé zij AI inzetten.



## Governance houdt de ontwikkeling niet bij

Slechts

**21%**

van de organisaties heeft een volwassen governancemodel voor autonome agents.[1] Zulke modellen leggen de grenzen, monitoring en auditability vast rond het werk dat agents uitvoeren..

Naarmate agents meer samenhangende taken uitvoeren, wordt deze kloof steeds belangrijker. Meer autonomie vraagt om duidelijkere verantwoordelijkheid, sterkere controles en vastgelegde validatieprocessen.

## De richting is duidelijk, de implementatie is nog in ontwikkeling

Agentic development krijgt een plek in echte development-workflows, maar adoptie alleen zegt niets over volwassenheid.

> Veel organisaties moeten eenvoudige AI-ondersteuning nog achter zich laten en de processen, controles en governance inrichten die nodig zijn voor verantwoord agentic development. De uitdaging is om groeiend gebruik om te zetten in consistente en meetbare waarde.


**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

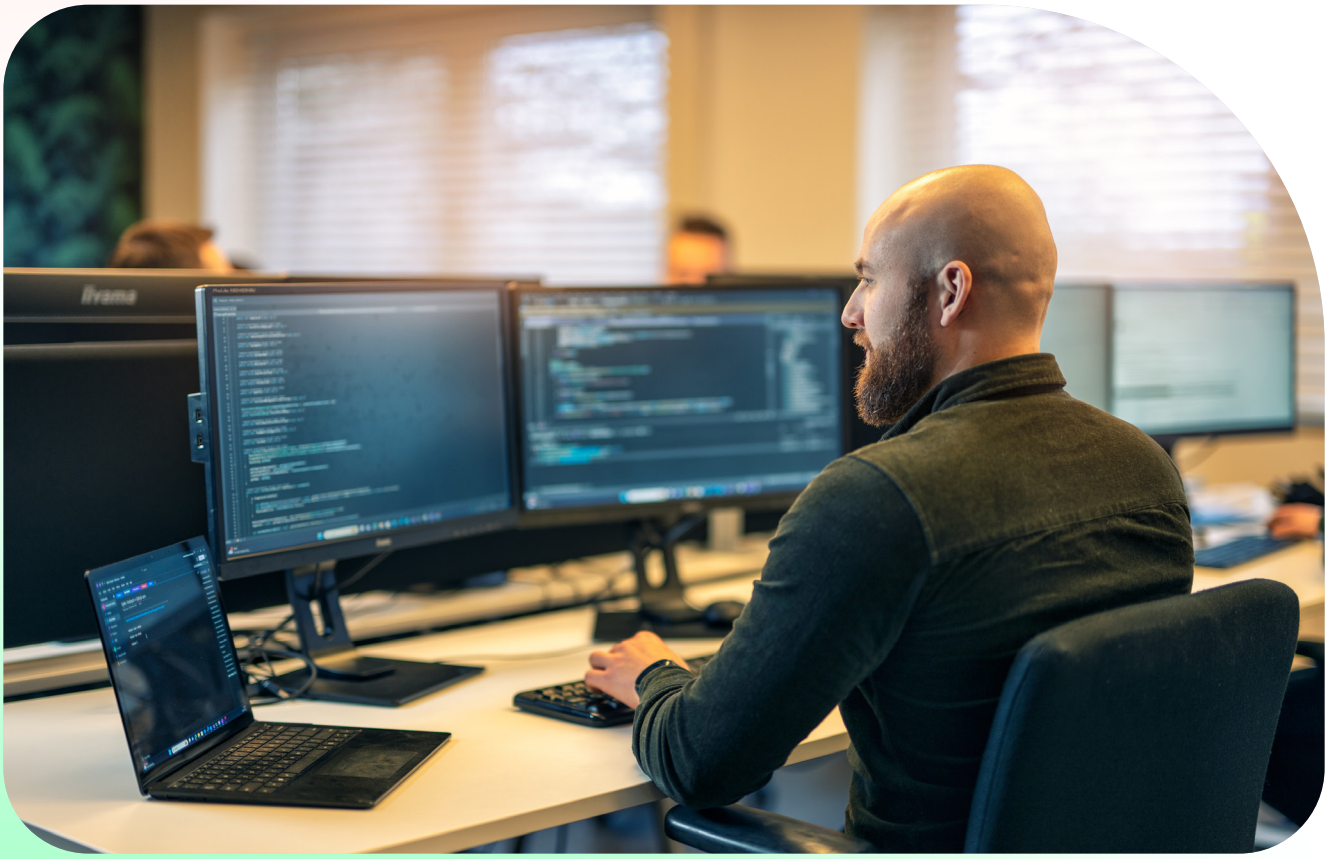
# 3. Agentic development in de praktijk

Agentic development verandert hoe softwareteams het werk verdelen en coördineren. Developers hoeven niet langer iedere stap zelf uit te voeren. Zij kunnen een doel bepalen en samenhangende developmenttaken aan agents delegeren.

Een team dat agentic development toepast, bestaat nog steeds uit mensen in rollen zoals product owner, engineer en QA- of operationsspecialist. Zij bepalen het werk, bieden context en delegeren specifieke taken aan agents.

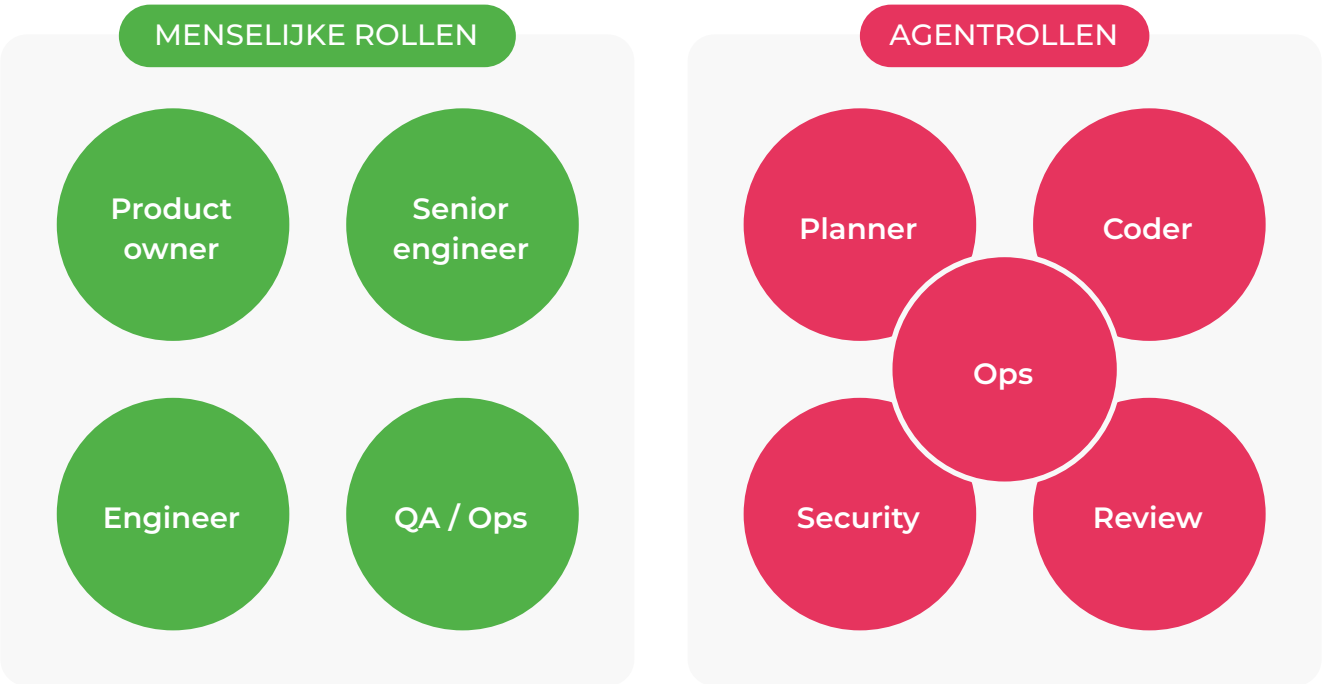
Ook agents kunnen **gespecialiseerde rollen** hebben. Zo kan een planner een aanpak voorstellen voordat een coding agent deze implementeert. Een security-agent controleert de wijzigingen op kwetsbaarheden, terwijl een reviewagent de kwaliteit en maintainability beoordeelt.

Zo ontstaat een workflow waarin iedere agent bijdraagt aan een afgebakend deel van het doel, terwijl developers het werk coördineren en het resultaat goedkeuren.



# Verantwoordelijkheden van mensen en agents

- AI-agents kunnen context verzamelen, een aanpak plannen, code wijzigen, tools gebruiken en reageren op de resultaten.
- Developers bepalen het doel, bieden context en stellen de grenzen vast waarbinnen agents mogen werken. Zij reviewen de output, nemen beslissingen die menselijk inzicht vragen en blijven verantwoordelijk voor het resultaat.






## De agentic development workflow

Een workflow voor agentic development bestaat uit vier verbonden fasen. De agent verzamelt context, plant het werk, implementeert de oplossing en reviewt het resultaat voordat een developer het beoordeelt:

| | | |
|---|---|---|
| 1 | **Research** | De agent onderzoekt het doel, de codebase, de beperkingen, dependencies en mogelijke risico's. |
| 2 | **Plan** | De agent bepaalt de aanpak, verdeelt het werk in taken, selecteert de benodigde tools en legt vast hoe het resultaat wordt gevalideerd. |
| 3 | **Implement** | De agent voert de geplande wijzigingen uit en draait doorlopend relevante tests en checks. |
| 4 | **Review** | De agent beoordeelt het resultaat op kwaliteit, security en juistheid en verbetert het werk voordat een developer het goedkeurt. |

De developer blijft betrokken als extra context of menselijk inzicht nodig is en keurt het eindresultaat goed.

## Zonder context moet een agent gokken

Een agent een taak geven zonder context is vergelijkbaar met werk toewijzen aan een nieuwe developer zonder de codebase of werkwijze uit te leggen. Voordat een agent een taak betrouwbaar kan uitvoeren, moet deze de projectstructuur, coding standards, architectuurbeslissingen en het verwachte resultaat begrijpen.

- Repository instructions beschrijven de coding conventions, projectstructuur en developmentpraktijken van het team.
- Skills bevatten herbruikbare instructies voor specifieke taken, zoals een API-endpoint maken of een securityreview uitvoeren.
- Model Context Protocol (MCP)-servers verbinden de agent met goedgekeurde tools en informatie buiten de directe codebase, zoals interne documentatie of testomgevingen.

Samen geven deze bronnen de agent de context om binnen de developmentomgeving van het team te werken.

# Agentic development binnen de SDLC

AI zet de traditionele SDLC op zijn kop en GitHub Copilot helpt dit mogelijk te maken. Neem een normale werkdag van een developer. Er komt een bugreport binnen. De developer onderzoekt het probleem, wijzigt de code, schrijft tests, opent een pull request, verwerkt feedback en monitort de pipeline. Veel stappen en veel contextwisselingen.

Met GitHub Copilot en agents houdt de mens de regie, terwijl agents meer werk tussen deze stappen uitvoeren. Dit zorgt voor een snellere, beter verbonden developmentworkflow zonder menselijke controle weg te nemen.

**GitHub Copilot Coding Agent** staat centraal in deze ontwikkeling. De agent kan een afgebakende taak van issue tot pull request uitvoeren, terwijl de developer verantwoordelijk blijft voor de review en goedkeuring van het resultaat.

Zes fases van de SDLC:




| Fase | Verantwoordelijkheid developer | Bijdrage agent |
|---|---|---|
| **Plan** | Bepaalt het doel, de scope en de acceptatiecriteria | Verzamelt context en stelt een implementatieplan voor |
| **Build** | Reviewt de aanpak en belangrijke ontwerpbeslissingen | Maakt een branch en implementeert de geplande wijzigingen |
| **Test** | Bepaalt wat moet worden gevalideerd | Genereert tests, voert checks uit en reageert op failures |
| **Review** | Beoordeelt de wijzigingen en keurt het resultaat goed. | Vat het werk samen en signaleert mogelijke problemen |
| **Secure** | Beslist hoe risico's worden behandeld | Identificeert kwetsbaarheden en stelt gerichte fixes voor |
| **Operate** | Bevestigt productie-impact | Analyseert pipelinefailures en stelt correcties voor |

Binnen de SDLC blijft de balans hetzelfde: developers geven richting en nemen de eindbeslissingen, terwijl agents afgebakend werk binnen die grenzen uitvoeren.

# 4. Wat agentic development oplevert

Voor technologieleiders wordt de waarde van agentic development zichtbaar in meetbare resultaten: kortere deliverycycli, betere kwaliteit en security, minder knelpunten en meer zicht op de voortgang.



Hoe ver een team is met de adoptie, bepaalt hoeveel waarde het kan behalen. Code completion ondersteunt afzonderlijke taken. Wanneer teams taken gaan delegeren, kunnen developers complete opdrachten aan agents toewijzen. Verder gevorderde teams kunnen meerdere gespecialiseerde agents binnen de SDLC coördineren. De winst strekt zich dan uit van individuele productiviteit tot teambrede delivery.

## Snellere delivery met minder knelpunten

Wanneer agents meer procedureel werk uitvoeren, besteden developers meer tijd aan beslissingen en minder aan het uitzoeken van details. Een bugfix kan worden geïmplementeerd en met tests gevalideerd voordat deze de main branch bereikt. Een pull request kan de wijzigingen en mogelijke risico's in duidelijke taal toelichten. Zo wordt het traject van issue naar goedgekeurde wijziging korter.

Volgens GitHub mergen actieve Copilot-gebruikers **3,3 keer** meer pull requests en ronden zij merges **2,4 keer** sneller af dan passieve gebruikers.[7] De exacte impact verschilt per team, use case en adoptieniveau.

Wanneer development sneller verloopt, kan het knelpunt zich verplaatsen. Productteams moeten mogelijk sneller beslissingen nemen en prioriteiten stellen om de extra engineeringcapaciteit effectief te benutten.

## Hogere kwaliteit en minder releaserisico

Testing en security worden onderdeel van de developmentworkflow, in plaats van afzonderlijke controles aan het einde. Hierdoor vinden teams defects en kwetsbaarheden eerder, wanneer ze eenvoudiger en goedkoper zijn op te lossen.

Geautomatiseerde checks kunnen routineproblemen vóór peer review identificeren. Menselijke reviewers richten zich daardoor op juistheid, securityrisico's en maintainability. Dit verkort reviewcycli, benut senior expertise beter en verkleint de kans dat kostbare problemen productie bereiken.

Het resultaat is snellere feedback, minder rework en meer vertrouwen in iedere release.

## Meer zicht en voorspelbaarheid

Het is essentieel om te weten wanneer werk klaar is, of een release veilig kan plaatsvinden en waar knelpunten ontstaan.

Een pull request bevat een samen-vatting van de wijzigingen.

Een dependency alert bevat een voorgestelde fix.

Een falende pipe-line toont de waar-schijnlijke oorzaak en een voorgestelde correctie.

Informatie waarvoor eerder extra meetings of onderzoek nodig waren, komt nu direct naast het werk beschikbaar. Zo krijgen teams een duidelijker beeld van de voortgang, risico's en release readiness.



## Meer tijd voor beslissingen en samenwerking

Wanneer agents routinetaken rond implementatie, testen en review overnemen, houden developers meer tijd over voor de onderdelen van softwareontwikkeling die overleg en menselijk inzicht vragen.

Zij kunnen nauwer met product managers, designers, testers en operationsspecialisten werken aan requirements, architectuur, edge cases en user experience. In plaats van vooral mechanische developmenttaken uit te voeren, bepalen ze wat gebouwd moet worden en of het resultaat aan de behoefte voldoet.

Dit versterkt de samenwerking binnen het developmentproces. Developers sturen agents steeds vaker aan en beoordelen hun output, terwijl zij hun eigen expertise inzetten waar die de meeste waarde toevoegt.

## Consistentere development op schaal

Agentic workflows kunnen gedeelde instructions, kwaliteitscontroles en goedkeuringsregels binnen verschillende teams toepassen. Hierdoor is het eenvoudiger om developmentpraktijken consistent te houden terwijl de adoptie groeit.

Duidelijke checkpoints houden developers in controle over beslissingen met een hoger risico. Agents voeren routinewerk uit binnen vastgelegde grenzen. Organisaties kunnen daardoor capaciteit toevoegen zonder toezicht of accountability op te geven.

De bedrijfswaarde gaat verder dan meer code produceren. Ze ontstaat door kwalitatieve software met minder knelpunten, duidelijke verantwoordelijkheid en meer tijd voor menselijke beslissingen door de SDLC te bewegen.

# 5. Praktijkvoorbeelden

De volgende scenario's laten zien hoe developers samenhangende taken aan GitHub Copilot Agents kunnen delegeren, terwijl zij verantwoordelijk blijven voor het resultaat.

SCENARIO 1

### Een complexe bug oplossen

Een productteam ontdekt een bug die alleen optreedt wanneer gebruikers snel tussen schermen wisselen. Een developer opent een GitHub Issue en wijst dit toe aan de Copilot Coding Agent. De agent maakt een branch, voegt test scaffolding toe en bepaalt waar applicatiestate mogelijk lekt.

In Visual Studio Code vraagt de developer GitHub Copilot de relevante statevariabele te traceren en een fix voor te stellen. Na review gebruikt de developer Playwright via een MCP-server om te bevestigen dat het probleem niet meer optreedt. Zo kan de developer de oorzaak en oplossing beoordelen in plaats van ieder component handmatig te traceren.

SCENARIO 2

### Een applicatie naar een andere taal migreren

Een team moet een actieve HTTP API van Python met FastAPI naar C# met .NET Minimal APIs migreren zonder het gedrag te veranderen.

Copilot Agent Mode analyseert de codebase, plant de architectuurwijzigingen en bouwt de endpoints opnieuw in het nieuwe framework. De agent zet datastructuren om, voegt Swagger- en OpenAPI-documentatie toe en maakt unit tests om beide applicaties te vergelijken.

De developer reviewt de architectuur en bevestigt dat de gemigreerde applicatie aan de oorspronkelijke requirements voldoet. De agent voert veel repetitief vertaal- en validatiewerk uit; de developer richt zich op compatibiliteit, ontwerp en kwaliteit.



SCENARIO 3

## Een defecte CI/CD-pipeline herstellen

Kort voor een geplande release faalt een GitHub Actions-build doordat een update van een externe dependency een breaking change introduceert. Verdere deployments zijn geblokkeerd.

Wanneer de build faalt, maakt een geconfigureerde workflow een issue met de relevante logs en wijst dit toe aan de GitHub Copilot Coding Agent. De agent analyseert de fout, identificeert de bijgewerkte dependency en bereidt een wijziging voor die deze vastzet op de laatste stabiele versie. Daarna opent de agent een draft pull request, waarin GitHub Actions de correctie valideert voordat de developer deze reviewt.

De developer beoordeelt de wijziging en beslist of deze kan worden gemerged. Het team krijgt een geteste correctie zonder meerdere engineers hetzelfde probleem te laten onderzoeken.

In alle scenario's blijft de werkverdeling gelijk. Developers bepalen het doel, bieden context en keuren het resultaat goed. Agents voeren het verbonden onderzoek, de implementatie en validatie tussen die menselijke checkpoints uit.
checkpoints.

# 6. Je team voorbereiden op agentic development

Agentic development vraagt om verandering binnen de organisatie én in het werk van developers. Succesvolle adoptie vereist consistente toegang, praktijkervaring en duidelijke grenzen.

## Start met een gerichte pilot

Kies één of twee workflows met een duidelijk doel en beperkt risico, zoals bugs onderzoeken, tests genereren, dependencies bijwerken of falende pipelines diagnosticeren.

Leg vooraf vast welk resultaat je verwacht, bijvoorbeeld kortere cyclustijden, minder reviewwerk of sneller herstel na failures. Met een afgebakende pilot kan het team de waarde bevestigen, ontbrekende context vinden en de controles verbeteren voordat de adoptie wordt uitgebreid.

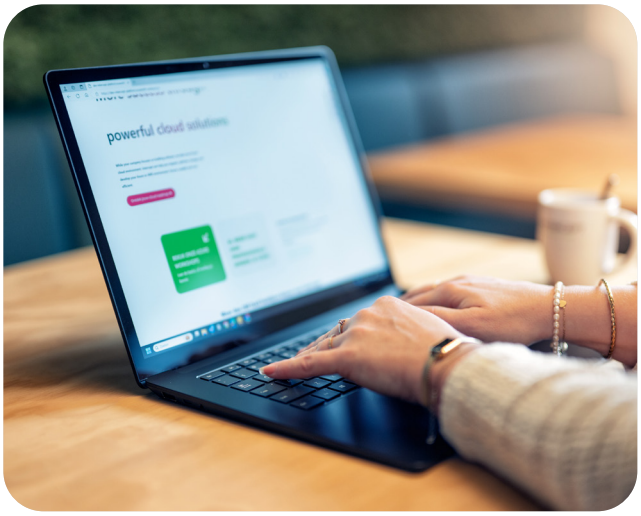
Teams hebben ook consistente toegang nodig tot de vereiste GitHub Copilot-functionaliteiten. Business- en Enterprise-abonnementen bieden de gedeelde functies en policy controls die nodig zijn zodra de adoptie verder gaat dan individuele experimenten.

## Bouw praktische vaardigheden en de juiste cultuur

Developers die code completion kennen, hebben mogelijk nog begeleiding nodig bij Agent Mode, Coding Agent, MCP-servers en het reviewen van agent-generated changes.

Door hands-on met de eigen codebase te oefenen, leren developers waar agents goed presteren en waar nauwer toezicht nodig is. Ook product owners, architecten, testers en operationsspecialisten moeten begrijpen hoe agents hun deel van de workflow beïnvloeden.

Leiders moeten GitHub Copilot positioneren als samenwerkingspartner, niet als vervanging of shortcut. Developers blijven verantwoordelijk voor het aansturen van agents, het reviewen van hun werk en het goedkeuren van het resultaat.






## Bied context en stel guardrails vast

Agents hebben duidelijke doelen, acceptatiecriteria en toegang tot de benodigde instructies en tools nodig. Repository instructions, skills en goedgekeurde MCP-servers helpen agents binnen de werkelijke developmentomgeving van het team te werken.

Teams moeten ook bepalen welke acties agents zelfstandig mogen uitvoeren en welke menselijke goedkeuring vragen. Policies moeten rechten, security, toegang, kwaliteitsstandaarden en verantwoord gebruik omvatten.

Duidelijke reviewmomenten, monitoring en audit trails maken zichtbaar wat een agent heeft gewijzigd en wanneer ingrijpen nodig is. Agents kunnen wijzigingen voorstellen en implementeren; teams blijven verantwoordelijk voor wat productie bereikt.

## Meet, leer en schaal op

Alleen het gebruik laat niet zien of de adoptie waarde oplevert. Meet daarom resultaten zoals cycle time, time to merge, reviewinspanning, escaped defects, pipeline recovery en developertevredenheid.

Gebruik deze resultaten om te bepalen waar agentic workflows waarde leveren en waar teams extra ondersteuning nodig hebben. Breid de adoptie geleidelijk uit over teams en use cases en pas daarbij de standaarden en lessen uit de pilot toe.



# Klaar voor agentic development met GitHub Copilot?

**Veel organisaties gebruiken GitHub Copilot al, maar niet iedere developer of ieder team haalt er evenveel waarde uit.**

Tijdens een interactieve CIE-sessie werken developers met praktische scenario's en leren zij hoe ze agentic development met GitHub Copilot effectiever toepassen in hun dagelijkse workflows.

VRAAG EEN CIE-SESSIE AAN

Georganiseerd door Microsoft, GitHub en Intercept.