Agentic Software Development

Entwicklungsaufgaben kontrolliert an AI-Agenten delegieren

01 INTRO

AI-Agenten können Entwicklungsaufgaben nicht nur unterstützen, sondern über mehrere Arbeitsschritte hinweg selbstständig bearbeiten. Sie analysieren Codebasen, planen Änderungen, bearbeiten Dateien, führen Werkzeuge aus, testen Ergebnisse und reagieren auf Fehler. Damit entstehen neue Möglichkeiten – aber auch neue Anforderungen an Kontrolle, Sicherheit und Qualitätssicherung.

Der Kurs vermittelt die technischen und methodischen Grundlagen für agentische Softwareentwicklung. Die Teilnehmer lernen, Entwicklungsaufgaben für Agenten zu strukturieren, Werkzeuge und Kontext bereitzustellen, Autonomie gezielt zu begrenzen und Ergebnisse durch Tests, Quality Gates und technische Kontrollmechanismen abzusichern. Der Schwerpunkt liegt auf produktneutralen Prinzipien, die sich auf unterschiedliche Coding Agents und Entwicklungsumgebungen übertragen lassen.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Praktische Erfahrung in der Softwareentwicklung, Git und automatisierten Tests

02 TRAININGSPROGRAMM

Inhalte

Vom AI-Assistenten zum Agenten

- AI-Assistent und AI-Agent
- Chat, Coding Assistant und Coding Agent
- Wahrnehmen, Entscheiden und Handeln
- mehrstufige Aufgaben
- Tool-Nutzung
- Feedback aus der Umgebung
- unterschiedliche Autonomiegrade
- geeignete Aufgaben für Agenten

Aufbau eines agentischen Entwicklungsprozesses

- Auftrag
- Kontext
- Planung
- Ausführung
- Beobachtung des Ergebnisses
- Verifikation
- Korrektur
- Abschlussbedingungen
- iterative Agent Loops

Entwicklungsaufträge für Agenten

- Ziel eindeutig definieren
- Ausgangszustand
- gewünschtes Verhalten
- Akzeptanzkriterien
- technische Randbedingungen
- Nicht-Ziele
- erlaubter Änderungsumfang
- Definition of Done

Aufgaben zerlegen und planen

- geeignete Größe agentischer Aufgaben
- Analyse vor Implementierung
- Abhängigkeiten erkennen
- Arbeitsschritte planen
- Zwischenziele definieren
- Plan gegen Anforderungen prüfen
- dynamische Plananpassung
- Abbruchbedingungen

Kontextmanagement

- Repository als Kontext
- Projektwissen
- Anforderungen und Dokumentation
- Architekturinformationen
- Coding Conventions
- relevante und irrelevante Informationen
- Kontextgrenzen
- Kontext über längere Aufgaben erhalten

Persistentes Projektwissen

- Projektinstruktionen
- Architekturregeln
- Entwicklungsregeln
- Build- und Testinformationen
- wiederverwendbares Wissen
- projektspezifische Agentenanweisungen
- Versionierung von Instruktionen
- Regeln knapp und wartbar halten

Tools als Grundlage agentischer Arbeit

- Agent ohne Tools und Agent mit Tools
- Dateisystem
- Shell
- Compiler und Interpreter
- Build-Systeme
- Testframeworks
- Git
- APIs und externe Systeme
- Browser und weitere Entwicklungswerkzeuge

Tool-Schnittstellen gestalten

- klar abgegrenzte Operationen
- strukturierte Parameter
- strukturierte Ergebnisse
- Fehlermeldungen
- Seiteneffekte
- idempotente Operationen
- minimale Tool-Oberflächen
- Tools für Agenten statt für Menschen gestalten

Model Context Protocol

- MCP als Tool- und Kontextschnittstelle
- MCP Server
- Tools
- Resources und weitere bereitgestellte Informationen
- vorhandene MCP-Server einsetzen
- eigene Entwicklungswerkzeuge anbinden
- Toolsets strukturieren
- MCP in agentischen Entwicklungsumgebungen

Berechtigungen und Least Privilege

- Lesen und Schreiben unterscheiden
- Arbeitsbereiche begrenzen
- Shell-Zugriff
- Netzwerkzugriff
- externe Systeme
- Credentials und Secrets
- minimale benötigte Rechte
- Berechtigungen abhängig von der Aufgabe

Kontrollierte Autonomie

- vollständig manuelle Freigabe
- Freigabe bestimmter Aktionen
- Policy-basierte Freigabe
- autonom ausführbare Aktionen
- kritische Aktionen
- reversible und irreversible Operationen
- Eskalation an den Menschen
- Autonomie nach Risiko festlegen

Human in the Loop

- sinnvolle Kontrollpunkte
- Plan vor Ausführung prüfen
- kritische Änderungen freigeben
- Zwischenstände beurteilen
- Unsicherheit eskalieren
- Entscheidungen beim Menschen belassen
- unnötige Freigaben vermeiden
- Kontrolle ohne Mikromanagement

Technische Guardrails

- Regeln außerhalb des Modells
- Allow- und Deny-Regeln
- Tool-Policies
- Hooks
- Wrapper und Gateways
- Dateisystemgrenzen
- Netzwerkgrenzen
- Validierung vor Aktionen
- Regeln technisch erzwingen

Isolation und Sandboxing

- Agenten vom Hostsystem begrenzen
- Container
- isolierte Arbeitsverzeichnisse
- temporäre Umgebungen
- eingeschränkte Credentials
- Netzwerkisolation
- reproduzierbare Entwicklungsumgebungen
- Isolation nach Risikoklasse

Git als Sicherheits- und Kontrollmechanismus

- sauberer Ausgangszustand
- Branches
- kleine Änderungen
- Diffs
- Commits als Checkpoints
- Änderungen zurücksetzen
- Historie nachvollziehen
- Agentenarbeit reversibel gestalten

Worktrees und parallele Arbeitsbereiche

- unabhängige Arbeitsbereiche
- Git Worktrees
- parallele Agenten
- Änderungen voneinander isolieren
- Branches pro Aufgabe
- Konflikte vermeiden
- Ergebnisse zusammenführen
- Aufräumen nach Abschluss

Tests als ausführbare Spezifikation

- Anforderungen in Tests überführen
- Tests vor Implementierung
- Akzeptanztests
- Regressionstests
- Grenz- und Fehlerfälle
- Tests als Feedback für Agenten
- Testqualität überprüfen
- Agent darf nicht einfach die Spezifikation passend machen

Quality Gates

- Build
- Compiler
- automatisierte Tests
- Linting
- statische Analyse
- Typprüfung
- Security Scans
- Architekturregeln
- verbindliche Kriterien vor Abschluss

Agentische Fehlerkorrektur

- fehlgeschlagene Builds
- fehlgeschlagene Tests
- Fehlermeldungen als Feedback
- Ursache analysieren
- erneuter Lösungsversuch
- Schleifen begrenzen
- wiederholte Fehler erkennen
- Eskalation statt Endlosschleife

Zustandsmanagement

- aktueller Arbeitsstand
- erledigte und offene Schritte
- Zwischenergebnisse
- Entscheidungen
- Git als persistenter Zustand
- externe Statusinformationen
- Fortsetzen nach Unterbrechungen
- Wiederaufnahme längerer Aufgaben

Lange laufende Entwicklungsaufgaben

- Aufgaben über längere Zeiträume
- Checkpoints
- Fortschrittsprotokolle
- Kontextverlust
- Wiederanlauf
- Teilresultate sichern
- Abbruch und Fortsetzung
- Abschluss eindeutig erkennen

Subagents

- spezialisierte Agenten
- Recherche
- Architektur
- Implementierung
- Tests
- Review
- Kontext voneinander trennen
- Ergebnisse an einen koordinierenden Agenten zurückgeben

Multi-Agent-Workflows

- mehrere Agenten koordinieren
- unabhängige Aufgaben parallelisieren
- Verantwortlichkeiten trennen
- gemeinsame Ressourcen
- Abhängigkeiten
- konkurrierende Änderungen
- Ergebnisse integrieren
- zusätzliche Komplexität gegen Nutzen abwägen

Agent als Entwickler und Agent als Reviewer

- unterschiedliche Rollen
- Implementierung und Review trennen
- unabhängiger Kontext
- Anforderungen erneut prüfen
- Tests kritisch beurteilen
- Architekturverletzungen erkennen
- Security Review
- Grenzen von AI-gegen-AI-Kontrolle

Untrusted Input und Prompt Injection

- externe Inhalte als Eingabe
- Quellcode und Kommentare
- Dokumentation
- Webseiten
- Issues und Tickets
- direkte und indirekte Prompt Injection
- Daten nicht automatisch als Instruktionen behandeln
- Prompt-Regeln allein sind keine Sicherheitsgrenze

Secrets und sensible Informationen

- Zugangsdaten
- API Keys
- Konfigurationen
- Produktionsdaten
- personenbezogene Daten
- Logs
- Secret Stores
- Zugriff nur bei tatsächlichem Bedarf

Externe Systeme

- Datenbanken
- Issue Tracker
- Git-Plattformen
- CI/CD
- Cloud-Dienste
- Produktivsysteme
- unterschiedliche Risikoklassen
- Schreibzugriffe besonders absichern

Observability und Audit

- ausgeführte Aktionen protokollieren
- Tool-Aufrufe
- Dateiänderungen
- Entscheidungen und Ergebnisse
- Kosten und Ressourcen
- Fehler
- nachvollziehbare Agentenläufe
- Informationen für spätere Analyse erhalten

Fehler- und Recovery-Strategien

- teilweise ausgeführte Aufgaben
- Tool-Fehler
- Netzwerkfehler
- inkonsistente Zustände
- Rollback
- Retry
- idempotente Wiederholung
- kontrollierter Abbruch

Kosten und Ressourcen kontrollieren

- Modellwahl nach Aufgabe
- Tokenverbrauch
- Kontextgröße
- Tool-Aufrufe
- parallele Agenten
- Endlosschleifen
- Zeit- und Kostenlimits
- wirtschaftliche Grenzen definieren

Agenten in CI/CD

- agentische Entwicklung und deterministische Pipeline trennen
- Agent erzeugt Änderung
- klassische Pipeline verifiziert
- Quality Gates bleiben verbindlich
- reproduzierbare Builds
- automatische Reviews
- Freigabeprozesse
- AI-Code erhält keinen Sonderstatus

Agentische Entwicklungsarchitektur entwerfen

- Agent
- Modell
- Kontext
- Tool Layer
- Policy Layer
- Entwicklungsumgebung
- Quality Layer
- externe Systeme
- Trust Boundaries
- Verantwortlichkeiten der Komponenten

Einen agentischen Workflow entwickeln

- Entwicklungsaufgabe auswählen
- Risiken klassifizieren
- Kontext definieren
- benötigte Tools bestimmen
- Berechtigungen festlegen
- Verifikationsmechanismen definieren
- Eskalationspunkte festlegen
- Workflow implementieren und testen

Praktische Übungen

- Entwicklungsaufgabe für einen Agenten formulieren
- Agentenauftrag in kontrollierbare Schritte zerlegen
- Toolset für die Aufgabe definieren
- Berechtigungen begrenzen
- Tests als ausführbare Anforderungen erstellen
- Quality Gates einrichten
- fehlgeschlagenen Agentenlauf analysieren
- Git und Worktrees zur Isolation einsetzen
- Subagent für eine abgegrenzte Aufgabe verwenden
- Prompt-Injection-Szenario untersuchen
- vollständigen agentischen Entwicklungsworkflow entwerfen und durchführen

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	3 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams
Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner mit Entwicklungsumgebung, Git und Zugang zu einem Coding Agent

> **Inhouse-Schulungen**
> Vorhandene Coding Agents, Repositories, CI/CD-Systeme, MCP-Server, Entwicklungswerkzeuge und unternehmensinterne Sicherheitsanforderungen können in die Übungen einbezogen werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an erfahrene Softwareentwickler, Softwarearchitekten, DevOps Engineers, Quality Engineers und technische Consultants, die Coding Agents nicht nur als persönliche Entwicklungshelfer einsetzen, sondern agentische Entwicklungsprozesse systematisch gestalten und kontrollieren möchten.

Er eignet sich insbesondere für Teams, die Entwicklungsaufgaben zunehmend an AI-Agenten delegieren und dafür belastbare technische Grenzen, Qualitätsmechanismen und Arbeitsabläufe etablieren wollen.

> **Voraussetzungen**
> Praktische Erfahrung in der Softwareentwicklung, Git und automatisierten Tests wird vorausgesetzt. Grundkenntnisse im Umgang mit generativer KI und AI-gestützter Softwareentwicklung sind hilfreich. Der Kurs setzt jedoch kein bestimmtes Coding-Agent-Produkt voraus.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- AI-Assistenten und AI-Agenten voneinander abgrenzen
- geeignete Entwicklungsaufgaben für Agenten identifizieren
- agentische Aufgaben mit klaren Zielen und Grenzen formulieren
- komplexe Aufgaben in kontrollierbare Arbeitsschritte zerlegen
- Kontext und persistentes Projektwissen bereitstellen
- geeignete Tools für Agenten auswählen und gestalten
- MCP als Tool-Schnittstelle einsetzen
- Berechtigungen nach dem Least-Privilege-Prinzip definieren
- Autonomie abhängig vom Risiko begrenzen
- technische Guardrails und Kontrollpunkte entwickeln
- Agenten durch Isolation und Sandboxing begrenzen
- Git und Worktrees als Kontrollmechanismen einsetzen
- Tests als ausführbare Spezifikation und Feedback verwenden
- verbindliche Quality Gates etablieren
- Subagents und Multi-Agent-Workflows einordnen
- Prompt-Injection- und Tool-Risiken berücksichtigen
- längere Agentenläufe beobachtbar und wiederaufnehmbar gestalten
- Fehler-, Recovery- und Eskalationsstrategien definieren
- einen produktneutralen agentischen Entwicklungsworkflow entwerfen und absichern

06 FAQ

Häufige Fragen

Was unterscheidet diesen Kurs von AI-assisted Software Development?

AI-assisted Software Development betrachtet den gesamten Entwicklungsprozess mit AI-Unterstützung. Der Entwickler führt dabei weiterhin den Prozess und verwendet KI für Analyse, Implementierung, Tests, Refactoring, Debugging und andere Aufgaben.

Dieser Kurs konzentriert sich dagegen auf Entwicklungsprozesse, in denen AI-Agenten mehrstufige Aufgaben selbstständig bearbeiten. Im Mittelpunkt stehen deshalb Tool-Nutzung, Berechtigungen, kontrollierte Autonomie, Guardrails, Isolation, Quality Gates, Zustandsmanagement und Orchestrierung.

Ist der Kurs auf Claude Code oder GitHub Copilot ausgerichtet?

Nein. Der Kurs behandelt produktneutrale Konzepte und Architekturen für agentische Softwareentwicklung. Konkrete Werkzeuge können für Übungen verwendet werden, sind aber nicht Gegenstand einer Produktschulung.

Werden Coding Agents praktisch eingesetzt?

Ja. Die Konzepte werden anhand praktischer Entwicklungsaufgaben erarbeitet. Entscheidend ist dabei nicht, möglichst viel Autonomie zu erreichen, sondern einen geeigneten und kontrollierbaren Autonomiegrad festzulegen.

Wird MCP behandelt?

Ja. MCP wird als eine Möglichkeit behandelt, Agenten strukturierte Werkzeuge und zusätzliche Informationsquellen bereitzustellen. Der Schwerpunkt liegt auf Tool-Grenzen, Berechtigungen und der Einordnung in die Gesamtarchitektur.

Werden Multi-Agent-Systeme behandelt?

Ja. Subagents und Multi-Agent-Workflows werden behandelt. Dabei wird auch untersucht, wann mehrere Agenten tatsächlich Vorteile bringen und wann sie lediglich zusätzliche Koordinationsprobleme erzeugen.

Geht es auch um Sicherheit?

Ja. Sicherheit ist ein wesentlicher Bestandteil agentischer Softwareentwicklung. Berechtigungen, Isolation, Secrets, externe Systeme, Prompt Injection und technisch erzwungene Guardrails werden deshalb ausführlich behandelt.

Warum reichen Prompt-Regeln für die Absicherung eines Agenten nicht aus?

Weil der Agent selbst Teil des zu kontrollierenden Systems ist. Kritische Grenzen sollten deshalb möglichst außerhalb des Modells durch Berechtigungen, Tool-Schnittstellen, Policies, Isolation und andere technische Mechanismen erzwungen werden.

Welche Rolle spielen Tests?

Eine zentrale. Tests können Anforderungen ausführbar machen und liefern einem Agenten objektives Feedback über seine Änderungen. Sie ersetzen jedoch weder die Prüfung der Testqualität noch weitere Quality Gates und Reviews.

Brauche ich AI-assisted Software Development vor diesem Kurs?

Nicht zwingend. Erfahrung mit AI-gestützter Softwareentwicklung ist hilfreich. Teilnehmer mit praktischer Entwicklungserfahrung und grundlegendem Verständnis generativer KI können direkt einsteigen.

Warum dauert der Kurs drei Tage?

Der Kurs verzichtet bewusst auf die breite Behandlung von Codegenerierung, Refactoring, Debugging und Dokumentation. Die drei Tage konzentrieren sich stattdessen auf Agentenarchitektur, Tools, Berechtigungen, Verifikation, Security und praktische agentische Workflows.

Kann der Kurs an unsere Entwicklungsumgebung angepasst werden?

Ja. Bei Inhouse-Schulungen können vorhandene Coding Agents, Repositories, CI/CD-Systeme, MCP-Server, Entwicklungswerkzeuge und unternehmensinterne Sicherheitsanforderungen in die Übungen einbezogen werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/agentic-software-development/