AI-assisted Software Development

Generative KI systematisch in der Softwareentwicklung einsetzen

01 INTRO

Generative KI kann heute in nahezu allen Phasen der Softwareentwicklung unterstützen – von der Analyse einer bestehenden Codebasis über Entwurf und Implementierung bis zu Tests, Dokumentation und Refactoring. Der produktive Einsatz geht dabei deutlich über das Erzeugen einzelner Codefragmente im Chat hinaus.

Der Kurs zeigt, wie Large Language Models und AI-Coding-Werkzeuge in einen professionellen Entwicklungsprozess integriert werden können. Die Teilnehmer arbeiten mit bestehenden und neuen Codebasen, formulieren Entwicklungsaufgaben für KI-Systeme, prüfen erzeugten Code und entwickeln Arbeitsweisen, bei denen Geschwindigkeit, Nachvollziehbarkeit und Softwarequalität zusammenpassen.

DAUER

4 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Praktische Erfahrung in der Softwareentwicklung und sicherer Umgang mit mindestens einer Programmiersprache

02 TRAININGSPROGRAMM

Inhalte

AI-assisted Software Development einordnen

- vom Code Completion zum AI Coding
- Chat, IDE-Assistent und Coding Agent
- unterschiedliche Formen der AI-Unterstützung
- typische Einsatzgebiete im Entwicklungsprozess
- Stärken und Schwächen aktueller Modelle
- Aufgaben nach Eignung auswählen
- Greenfield und bestehende Systeme
- Entwicklerrolle und Verantwortung

Entwicklungsumgebung und Werkzeuge

- AI-Funktionen in Entwicklungsumgebungen
- Chat-basierte Coding-Werkzeuge
- agentische Coding-Werkzeuge
- CLI-basierte Werkzeuge
- Zugriff auf Projektdateien und Repository
- Terminal und Entwicklungswerkzeuge
- Modell- und Werkzeugauswahl
- lokale und cloudbasierte Ansätze

Bestehende Codebasen erschließen

- Repository-Struktur analysieren
- relevante Dateien identifizieren
- Architektur und Abhängigkeiten erfassen
- vorhandene Konventionen erkennen
- Einstiegspunkte und Datenflüsse untersuchen
- unbekannten Code erklären lassen
- gezielte Fragen an die Codebasis
- Modellannahmen gegen den tatsächlichen Code prüfen

Kontext für Entwicklungsaufgaben

- benötigten Kontext bestimmen
- Code, Dokumentation und Anforderungen kombinieren
- relevante Dateien auswählen
- unnötigen Kontext vermeiden
- Projektregeln und Konventionen bereitstellen
- Architekturentscheidungen berücksichtigen
- Kontextgrenzen
- Kontext während längerer Aufgaben erhalten

Anforderungen in Entwicklungsaufgaben übersetzen

- fachliche Anforderung und technische Aufgabe unterscheiden
- Akzeptanzkriterien
- Randbedingungen
- vorhandenes Verhalten
- gewünschtes Verhalten
- explizite Nicht-Ziele
- technische Constraints
- Definition of Done

Aufgaben sinnvoll zerlegen

- geeignete Größe einer AI-Coding-Aufgabe
- große Änderungen in Arbeitsschritte zerlegen
- Abhängigkeiten zwischen Schritten
- Analyse vor Implementierung
- Planung durch das Modell
- Zwischenstände prüfen
- kontrollierte Iterationen
- Änderungen begrenzen

Code mit AI entwickeln

- neuen Code erzeugen
- bestehende Funktionen erweitern
- Boilerplate und wiederkehrende Strukturen
- APIs und Schnittstellen implementieren
- Datenverarbeitung
- Fehlerbehandlung
- bestehende Patterns übernehmen
- erzeugten Code verstehen statt nur übernehmen

AI und Softwaredesign

- Designvorschläge erzeugen und vergleichen
- Verantwortlichkeiten und Schnittstellen
- Kopplung und Kohäsion
- Abstraktionen
- bestehende Architektur respektieren
- unnötige Abstraktionen erkennen
- Overengineering durch AI
- Designentscheidungen beim Entwickler belassen

Refactoring mit AI

- Refactoring-Kandidaten identifizieren
- Code Smells untersuchen
- kleine und kontrollierte Refactorings
- Funktionen und Klassen zerlegen
- Duplikate reduzieren
- Namen und Strukturen verbessern
- Verhalten erhalten
- Refactoring gegen Tests absichern

AI-assisted Testing

- Testfälle aus Anforderungen ableiten
- Unit Tests erzeugen
- Parametrisierung
- Grenz- und Fehlerfälle
- bestehende Tests erweitern
- Testdaten erzeugen
- Testcode kritisch prüfen
- Tests nicht lediglich an die Implementierung anpassen

Test-first und TDD mit AI

- Test als ausführbare Anforderung
- Red-Green-Refactor mit AI-Unterstützung
- Tests vor Implementierung erzeugen
- Akzeptanzkriterien in Tests überführen
- AI auf einen kleinen Lösungsraum begrenzen
- Regressionen früh erkennen
- Grenzen AI-generierter Tests
- unabhängige Qualitätskontrolle erhalten

Fehleranalyse und Debugging

- Fehlermeldungen und Stacktraces analysieren
- Hypothesen erzeugen
- Fehler systematisch eingrenzen
- Logging und Diagnoseinformationen nutzen
- reproduzierbare Fehlerbeschreibungen
- minimale Reproduktionsfälle
- vorgeschlagene Fixes überprüfen
- Symptomkorrektur von Ursachenbehebung unterscheiden

Code Review mit AI

- Änderungen erklären lassen
- Diffs analysieren
- potenzielle Fehler identifizieren
- Architektur- und Stilregeln prüfen
- Security-Aspekte untersuchen
- fehlende Tests erkennen
- Review-Vorschläge verifizieren
- AI als zusätzliche Review-Instanz statt als Freigabeinstanz

Dokumentation mit AI

- bestehende Dokumentation aktualisieren
- README und technische Dokumentation
- API-Dokumentation
- Docstrings
- Architekturentscheidungen beschreiben
- Beispiele erzeugen
- Dokumentation gegen Code prüfen
- veraltete oder erfundene Informationen erkennen

Arbeiten mit Git und Versionskontrolle

- Änderungen überschaubar halten
- Diffs als Kontrollinstrument
- Commits sinnvoll strukturieren
- AI-generierte Änderungen vor Commit prüfen
- Branches und experimentelle Änderungen
- rücksetzbare Arbeitsschritte
- Commit Messages unterstützen
- Repository-Historie als Informationsquelle

Abhängigkeiten und externe Bibliotheken

- Bibliotheken und Frameworks auswählen
- APIs und Dokumentation berücksichtigen
- Versionsabhängigkeiten
- veraltete Modellkenntnisse
- nicht existente APIs und Funktionen
- Dependencies kritisch prüfen
- unnötige neue Abhängigkeiten vermeiden
- Herstellerdokumentation als Referenz

Halluzinationen beim Coding

- erfundene APIs
- falsche Parameter und Signaturen
- nicht vorhandene Dateien oder Funktionen
- falsche Annahmen über die Codebasis
- scheinbar plausible Implementierungen
- veraltetes Wissen
- Ergebnisse gegen Compiler, Interpreter und Tests prüfen
- ausführbare Verifikation statt Vertrauen

Security beim AI Coding

- sicherheitsrelevanten Code besonders behandeln
- Secrets und Zugangsdaten
- Eingabevalidierung
- Authentifizierung und Autorisierung
- Dependency-Risiken
- unsichere Codebeispiele
- externe Inhalte und Prompt Injection
- menschliches Review bei kritischen Änderungen

Datenschutz und vertraulicher Code

- welche Projektinformationen an Modelle übertragen werden
- Quellcode als vertrauliche Information
- personenbezogene Daten
- Konfigurationen und Logs
- Cloud- und lokale Modelle
- Unternehmensrichtlinien
- Zugriffsrechte
- geeignete Arbeitsumgebungen auswählen

Coding Agents

- Unterschied zwischen Assistent und Agent
- Dateien selbstständig untersuchen
- Dateien verändern
- Tests und Build-Werkzeuge ausführen
- Terminal und Tools verwenden
- mehrstufige Aufgaben bearbeiten
- Autonomiegrade
- Kontrolle über Änderungen behalten

Agenten kontrollieren

- Arbeitsbereich begrenzen
- Berechtigungen
- erlaubte und nicht erlaubte Aktionen
- Review vor kritischen Aktionen
- Shell- und Tool-Zugriffe
- externe Systeme
- nachvollziehbare Arbeitsschritte
- Zero-Trust-Prinzipien für agentische Entwicklung

Projektregeln für Coding Agents

- Coding Conventions
- Architekturregeln
- Testanforderungen
- Projektstruktur
- Build- und Testkommandos
- Definition of Done
- persistente Projektinstruktionen
- Regeln als Unterstützung, nicht als Sicherheitsgrenze

Tool-Nutzung und erweiterte Entwicklungsumgebungen

- Compiler und Interpreter
- Testframeworks
- Linter und Formatter
- statische Analyse
- Browser und UI-Automatisierung
- Datenbanken und APIs
- Dokumentation und Suche
- Werkzeuge gezielt für Verifikation einsetzen

Model Context Protocol und Tool-Anbindung

- MCP als Schnittstelle zu Werkzeugen und Kontext
- MCP Server und Tools
- Entwicklungswerkzeuge verfügbar machen
- strukturierte Tool-Aufrufe
- eigene Werkzeuge integrieren
- Berechtigungen und Grenzen
- Vertrauen in Tool-Ergebnisse
- MCP im Entwicklungsworkflow einordnen

Qualitätssicherung für AI-generierten Code

- AI-generierten Code wie fremden Code behandeln
- Compiler und statische Analyse
- automatisierte Tests
- Code Review
- Architekturregeln
- Security-Prüfungen
- Qualitäts-Gates
- unabhängige Verifikation

AI Coding in CI/CD

- AI während der Entwicklung und klassische Pipeline unterscheiden
- reproduzierbare Builds
- Tests als verbindliche Qualitätsgrenze
- Linting und statische Analyse
- Security Scans
- Qualitäts-Gates
- AI-generierte Änderungen durch die normale Pipeline führen
- keine Sonderbehandlung für AI-Code

Kosten und Effizienz

- Modellwahl nach Aufgabe
- Kontextgröße und Kosten
- große Codebasen
- wiederholte Analysen vermeiden
- kleine Modelle für geeignete Aufgaben
- Geschwindigkeit gegen Qualität abwägen
- Tokenverbrauch als technischer Faktor
- wirtschaftlichen Nutzen beurteilen

Ein professioneller AI-Coding-Workflow

- Anforderung verstehen
- Codebasis analysieren
- Änderung planen
- Aufgabe begrenzen
- Tests und Akzeptanzkriterien definieren
- Implementierung durchführen
- automatisch verifizieren
- Diff und Ergebnis reviewen
- dokumentieren und committen

Praktische Übungen

- bestehendes Repository mit AI erschließen
- Entwicklungsaufgabe formulieren
- Änderung planen und zerlegen
- Code mit AI implementieren
- Tests vor und nach der Implementierung entwickeln
- Fehler mit AI analysieren
- Refactoring durchführen
- AI-generierte Änderungen reviewen
- Coding Agent kontrolliert einsetzen
- vollständigen AI-assisted Entwicklungsworkflow durchführen

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	4 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 aktuellen Sprachmodell

> **Inhouse-Schulungen**
> Programmiersprachen, Repositories, Entwicklungswerkzeuge, Testframeworks und vorhandene AI-Coding-Werkzeuge können in die Übungen einbezogen werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Softwareentwickler, Softwarearchitekten, technische Consultants und andere Personen, die generative KI professionell in der Softwareentwicklung einsetzen möchten.

Er eignet sich sowohl für Entwickler, die bisher hauptsächlich Code Completion oder Chat-basierte Werkzeuge verwenden, als auch für Teilnehmer, die Coding Agents und stärker automatisierte Entwicklungsworkflows einsetzen möchten.

> **Voraussetzungen**
> Praktische Erfahrung in der Softwareentwicklung und sicherer Umgang mit mindestens einer Programmiersprache werden vorausgesetzt. Grundkenntnisse in Git, automatisierten Tests und typischen Entwicklungswerkzeugen sind hilfreich. Spezielle Vorkenntnisse zu Large Language Models oder Coding Agents sind nicht erforderlich.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- geeignete Aufgaben für AI-assisted Software Development identifizieren
- bestehende Codebasen mit AI-Unterstützung systematisch erschließen
- Entwicklungsaufgaben mit geeignetem Kontext und klaren Randbedingungen formulieren
- komplexe Änderungen in kontrollierbare Arbeitsschritte zerlegen
- AI für Implementierung, Refactoring, Testing, Debugging und Dokumentation einsetzen
- AI-generierten Code systematisch überprüfen

- Tests und TDD als Leitplanken für AI Coding einsetzen
- typische Halluzinationen und Fehler beim AI Coding erkennen
- Security- und Datenschutzaspekte berücksichtigen
- Coding Agents kontrolliert einsetzen
- Berechtigungen und Tool-Zugriffe angemessen begrenzen
- MCP und andere Tool-Anbindungen einordnen
- automatisierte Qualitätsmechanismen zur Verifikation einsetzen
- AI-generierte Änderungen in bestehende Entwicklungs- und CI/CD-Prozesse integrieren
- einen nachvollziehbaren und kontrollierten AI-Coding-Workflow etablieren

06 FAQ

Häufige Fragen

Für welche Programmiersprachen ist der Kurs geeignet?

Die grundlegenden Arbeitsweisen sind weitgehend unabhängig von einer bestimmten Programmiersprache. Übungen können an die im Unternehmen eingesetzten Sprachen und Technologien angepasst werden.

Ist der Kurs ein Training für ein bestimmtes AI-Coding-Produkt?

Nein. Werkzeuge können praktisch eingesetzt und verglichen werden, der Kurs ist jedoch nicht als Produktschulung konzipiert. Im Mittelpunkt stehen übertragbare Arbeitsweisen für AI-assisted Software Development.

Werden Coding Agents behandelt?

Ja. Der Kurs geht über einfache Code Completion und Chat-Nutzung hinaus. Agentische Werkzeuge, Tool-Zugriffe, Berechtigungen und kontrollierte autonome Arbeitsschritte sind Bestandteil des Trainings.

Werden auch Tests und TDD behandelt?

Ja. Tests sind ein zentraler Bestandteil des Kurses. Gerade bei AI-generiertem Code dienen sie als ausführbare Anforderungen und unabhängige Qualitätskontrolle.

Wird MCP behandelt?

Ja. Das Model Context Protocol wird im Zusammenhang mit der Anbindung von Entwicklungswerkzeugen und eigenen Tools behandelt. Der Schwerpunkt liegt auf der praktischen Einordnung und Nutzung im Entwicklungsworkflow.

Geht es auch um Sicherheit von Coding Agents?

Ja. Berechtigungen, Tool-Zugriffe, sensible Daten, Prompt Injection und die Begrenzung autonomer Aktionen werden behandelt.

Muss ich bereits Erfahrung mit AI-Coding-Werkzeugen haben?

Nein. Erfahrung mit ChatGPT, Claude, GitHub Copilot oder ähnlichen Werkzeugen ist hilfreich, aber nicht erforderlich. Softwareentwicklungserfahrung wird dagegen vorausgesetzt.

Warum dauert der Kurs vier Tage?

Der Kurs behandelt nicht nur das Erzeugen von Code. Die Teilnehmer arbeiten praktisch mit Codebasen, Tests, Refactoring, Debugging, Reviews, Coding Agents und Qualitätsmechanismen. Für diese Übungen und einen vollständigen AI-assisted Entwicklungsworkflow sind vier Tage realistisch.

Kann der Kurs an unsere Entwicklungsumgebung angepasst werden?

Ja. Bei Inhouse-Schulungen können Programmiersprachen, Repositories, Entwicklungswerkzeuge, Testframeworks und vorhandene AI-Coding-Werkzeuge in die Übungen einbezogen werden.

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