Claude Code im Einsatz

Claude Code produktiv und kontrolliert in der Softwareentwicklung einsetzen

01 INTRO

Claude Code arbeitet nicht nur mit einzelnen Codeausschnitten, sondern kann eine Codebasis untersuchen, Dateien verändern, Entwicklungswerkzeuge aufrufen, Tests ausführen und mehrstufige Aufgaben weitgehend selbstständig bearbeiten. Damit verschiebt sich die Arbeit mit generativer KI vom reinen Coding-Assistenten hin zu einem agentischen Entwicklungswerkzeug.

Der Kurs vermittelt den praktischen Einsatz von Claude Code in realen Softwareprojekten. Die Teilnehmer lernen, bestehende Codebasen zu erschließen, Entwicklungsaufgaben sinnvoll zu formulieren, Projektwissen bereitzustellen und Claude Code mit Entwicklungswerkzeugen und MCP-Servern zu verbinden. Ein besonderer Schwerpunkt liegt auf kontrollierter Autonomie, nachvollziehbaren Änderungen und der systematischen Verifikation der erzeugten Ergebnisse.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Praktische Erfahrung in der Softwareentwicklung, Git und im Umgang mit der Kommandozeile

02 TRAININGSPROGRAMM

Inhalte

Claude Code einordnen

- vom Chat zum Coding Agent
- agentische Softwareentwicklung
- Claude Code und die Claude-Modelle
- Terminal-basierte Arbeitsweise
- Codebasis als Arbeitskontext
- Dateien, Shell und Entwicklungswerkzeuge
- mehrstufige Aufgaben
- geeignete und ungeeignete Einsatzgebiete

Installation und Arbeitsumgebung

- Claude Code installieren
- Authentifizierung
- Projekt starten
- Arbeitsverzeichnis und Repository
- grundlegende Bedienung
- Sitzungen
- Modelle und Modellwahl
- Konfiguration

Eine bestehende Codebasis erschließen

- Repository untersuchen
- Verzeichnisstruktur verstehen
- relevante Dateien identifizieren
- Einstiegspunkte finden
- Abhängigkeiten nachvollziehen
- bestehende Patterns erkennen
- Architektur untersuchen
- Aussagen gegen den tatsächlichen Code prüfen

Aufgaben für Claude Code formulieren

- Ziel einer Änderung beschreiben
- Ausgangssituation bereitstellen
- gewünschtes Verhalten
- Akzeptanzkriterien
- technische Randbedingungen
- Nicht-Ziele
- Definition of Done
- Claude zunächst untersuchen statt sofort ändern lassen

Kontext gezielt bereitstellen

- vorhandener Repository-Kontext
- relevante Dateien
- Dokumentation
- Anforderungen
- Architekturinformationen
- Kontextumfang begrenzen
- fehlende Informationen erkennen
- Kontext über längere Aufgaben erhalten

CLAUDE.md und Projektwissen

- dauerhafte Projektinstruktionen
- Projektstruktur dokumentieren
- Coding Conventions
- Architekturregeln
- Build- und Testkommandos
- Qualitätsanforderungen
- wichtige Besonderheiten der Codebasis
- Projektwissen knapp und wartbar halten

Gute Projektinstruktionen entwickeln

- relevante von unnötigen Regeln unterscheiden
- konkrete statt allgemeiner Vorgaben
- vorhandene Dokumentation referenzieren
- Regeln für Implementierung
- Regeln für Tests
- Regeln für Änderungen
- Regeln für Verifikation
- Instruktionen als Hilfsmittel statt Sicherheitsgrenze

Entwicklungsaufgaben planen

- Analyse vor Implementierung
- vorhandenen Code lesen
- Lösungswege untersuchen
- Auswirkungen abschätzen
- größere Aufgaben zerlegen
- Plan überprüfen
- Implementierung begrenzen
- Zwischenstände kontrollieren

Code mit Claude Code entwickeln

- neue Funktionen implementieren
- bestehenden Code erweitern
- mehrere Dateien verändern
- bestehende Patterns übernehmen
- Schnittstellen implementieren
- Fehlerbehandlung
- Abhängigkeiten berücksichtigen
- erzeugten Code verstehen und überprüfen

Refactoring

- Refactoring-Kandidaten identifizieren
- Code Smells untersuchen
- Funktionen und Klassen zerlegen
- Duplikate reduzieren
- Verantwortlichkeiten verbessern
- größere Refactorings planen
- Verhalten durch Tests absichern
- Änderungen schrittweise durchführen

Tests mit Claude Code

- vorhandene Tests analysieren
- Testfälle aus Anforderungen ableiten
- Unit Tests entwickeln
- Integrations- und Systemtests ergänzen
- Grenzfälle
- Fehlerfälle
- Tests ausführen und Ergebnisse analysieren
- Testcode ebenso kritisch prüfen wie Produktivcode

Test-first und TDD

- Tests als ausführbare Anforderungen
- Akzeptanzkriterien in Tests überführen
- Test vor Implementierung
- Red-Green-Refactor
- Claude auf einen definierten Lösungsraum begrenzen
- Regressionen erkennen
- Tests nicht nachträglich passend machen
- unabhängige Qualitätskontrolle erhalten

Fehleranalyse und Debugging

- Fehlermeldungen untersuchen
- Stacktraces analysieren
- Hypothesen bilden
- relevante Codebereiche suchen
- reproduzierbare Fehlerfälle erstellen
- Logging einsetzen
- Fix entwickeln
- Ursache und Symptom unterscheiden

Shell und Entwicklungswerkzeuge

- Shell-Kommandos als Teil des agentischen Workflows
- Compiler und Interpreter
- Build-Werkzeuge
- Testframeworks
- Linter und Formatter
- statische Analyse
- Git
- eigene Projektwerkzeuge

Tool-Nutzung kontrollieren

- Lesen und Schreiben von Dateien
- Shell-Kommandos
- externe Werkzeuge
- Auswirkungen eines Tool-Aufrufs
- reversible und irreversible Aktionen
- Berechtigungen
- Freigaben
- Autonomie bewusst festlegen

Berechtigungsmodell und Sicherheit

- unterschiedliche Aktionsklassen
- Lese- und Schreibzugriffe
- Shell-Zugriffe
- Netzwerk und externe Systeme
- sensible Dateien
- Secrets
- kritische Aktionen
- Least Privilege und Zero Trust

Hooks

- Aktionen vor und nach Tool-Aufrufen
- wiederkehrende Prüfungen automatisieren
- Formatierung
- Tests
- Logging
- Policy Checks
- Aktionen zulassen oder verhindern
- Hooks als technische Kontrollschicht
- Projektregeln technisch absichern

Skills und wiederverwendbare Arbeitsweisen

- wiederkehrende Aufgaben standardisieren
- projektspezifisches Wissen kapseln
- Entwicklungsabläufe beschreiben
- Analyseaufgaben
- Test- und Review-Aufgaben
- Skills strukturiert aufbauen
- Teamwissen wiederverwenden
- Wartbarkeit wiederverwendbarer Instruktionen

Subagents

- Aufgaben an spezialisierte Agenten delegieren
- Kontext trennen
- Analyse und Implementierung aufteilen
- Reviewer als eigener Agent
- Rechercheaufgaben
- parallele Arbeit
- Ergebnisse zusammenführen
- unnötige Agentenkomplexität vermeiden

Model Context Protocol

- MCP einordnen
- MCP Server anbinden
- verfügbare Tools untersuchen
- eigene Entwicklungswerkzeuge bereitstellen
- APIs
- Datenbanken
- Browser und Testautomatisierung
- interne Werkzeuge

Eigene MCP-Werkzeuge

- geeignete Tool-Grenzen
- strukturierte Ein- und Ausgaben
- Tool-Beschreibungen
- Fehlerbehandlung
- minimale Berechtigungen
- vertrauenswürdige und nicht vertrauenswürdige Ergebnisse
- Tools testen
- Toolsets für Entwicklungsprojekte strukturieren

Claude Code und Browser-Automatisierung

- Browser als Entwicklungswerkzeug
- Webanwendungen untersuchen
- UI-Verhalten prüfen
- Screenshots und DOM-Informationen
- Playwright einsetzen
- Tests erzeugen und ausführen
- Fehler reproduzieren
- Entwicklung und UI-Verifikation verbinden

Git als Kontrollinstrument

- Änderungen über Diffs prüfen
- kleine Arbeitsschritte
- Commits als Checkpoints
- Branches
- unerwünschte Änderungen erkennen
- Arbeit zurücksetzen
- Repository-Historie nutzen
- agentische Arbeit nachvollziehbar halten

Code Review mit Claude Code

- Diffs analysieren
- Implementierungen erklären lassen
- potenzielle Fehler suchen
- fehlende Tests identifizieren
- Architekturregeln prüfen
- Security-Aspekte untersuchen
- Review von Implementierung trennen
- AI-Review nicht mit Freigabe gleichsetzen

Halluzinationen und Fehlannahmen

- erfundene APIs
- nicht vorhandene Dateien
- falsche Annahmen über Architektur
- veraltete Bibliothekskenntnisse
- scheinbar erfolgreiche Änderungen
- Tests, die das falsche Verhalten bestätigen
- Aussagen gegen Repository und Dokumentation prüfen
- ausführbare Verifikation bevorzugen

Lange und komplexe Aufgaben

- Aufgaben sinnvoll zerlegen
- Fortschritt dokumentieren
- Zwischenstände speichern
- Git als Zustandsspeicher
- Kontextgrenzen berücksichtigen
- Arbeit über mehrere Sitzungen
- Aufgabenstatus erhalten
- kontrolliert weiterarbeiten

Parallelisierung

- unabhängige Aufgaben erkennen
- parallele Analysen
- Subagents
- Konflikte vermeiden
- Ressourcenverbrauch berücksichtigen
- parallele Tool-Aufrufe
- Ergebnisse zusammenführen
- wann sequenzielles Arbeiten sinnvoller ist

Qualitätssicherung

- AI-generierten Code wie fremden Code behandeln
- Build
- Compiler
- automatisierte Tests
- statische Analyse
- Linter
- Security Checks
- Review
- verbindliche Quality Gates

Claude Code in bestehenden Entwicklungsprozessen

- lokale Entwicklung
- Issue-basierte Aufgaben
- Branch-Workflow
- Pull Requests
- CI/CD
- bestehende Qualitätsprozesse
- Teamregeln
- AI-Unterstützung ohne Sonderweg für AI-Code

Kosten und Ressourcen

- Modellwahl
- Kontextgröße
- Tokenverbrauch
- lange Sessions
- wiederholte Repository-Analysen
- parallele Agenten
- kleine und große Modelle
- Aufwand und Nutzen gegeneinander abwägen

Ein professioneller Claude-Code-Workflow

- Anforderung verstehen
- Repository untersuchen
- relevante Regeln und Dokumentation lesen
- Änderung planen
- Aufgabe begrenzen
- Tests und Akzeptanzkriterien definieren
- Implementierung durchführen
- Werkzeuge zur Verifikation einsetzen
- Diff überprüfen
- Review durchführen
- Ergebnis dokumentieren und committen

Praktische Übungen

- bestehendes Repository mit Claude Code erschließen
- CLAUDE.md für ein Projekt entwickeln
- Entwicklungsaufgabe formulieren und planen
- mehrstufige Änderung implementieren
- Tests entwickeln und ausführen
- Fehler analysieren und beheben
- Refactoring durchführen
- Hooks für automatische Prüfungen einsetzen
- MCP-Server anbinden
- externes Entwicklungswerkzeug über MCP verwenden
- Subagent für eine abgegrenzte Aufgabe einsetzen
- vollständigen Claude-Code-Workflow 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, Kommandozeile und Zugang zu Claude Code

> **Inhouse-Schulungen**
> Vorhandene Repositories, Programmiersprachen, Build- und Testwerkzeuge, CI/CD, MCP-Server und interne Entwicklungswerkzeuge können in die Übungen einbezogen werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Softwareentwickler, Softwarearchitekten, DevOps Engineers, Quality Engineers und technische Consultants, die Claude Code professionell für Softwareentwicklung und verwandte Engineering-Aufgaben einsetzen möchten.

Er eignet sich sowohl für Entwickler mit ersten Erfahrungen mit Claude Code als auch für Teams, die über einfache Coding-Unterstützung hinaus agentische Entwicklungsworkflows, MCP, Hooks und kontrollierte Automatisierung einsetzen möchten.

> **Voraussetzungen**
> Praktische Erfahrung in der Softwareentwicklung und sicherer Umgang mit mindestens einer Programmiersprache werden vorausgesetzt. Grundkenntnisse in Git und der Kommandozeile sind erforderlich, Erfahrung mit automatisierten Tests ist hilfreich. Vorkenntnisse zu Claude Code, MCP oder agentischen Systemen sind nicht erforderlich.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- Claude Code für professionelle Entwicklungsaufgaben einsetzen
- bestehende Codebasen systematisch erschließen
- Aufgaben mit geeignetem Kontext und klaren Randbedingungen formulieren
- CLAUDE.md und Projektinstruktionen sinnvoll strukturieren
- Entwicklungsaufgaben planen und kontrolliert umsetzen
- Code, Tests und Refactorings mit Claude Code entwickeln
- Shell und Entwicklungswerkzeuge in agentische Workflows integrieren
- Berechtigungen und Tool-Zugriffe angemessen begrenzen
- Hooks für automatische Kontrollen einsetzen
- wiederverwendbare Skills und Arbeitsweisen entwickeln
- Subagents für abgegrenzte Aufgaben einsetzen
- MCP-Server und eigene Tools integrieren
- Browser- und Testautomatisierung in den Workflow einbinden

- AI-generierten Code systematisch verifizieren
- Git als Kontroll- und Sicherungsmechanismus einsetzen
- längere agentische Aufgaben strukturiert bearbeiten
- Security-, Qualitäts- und Kostenaspekte berücksichtigen
- einen nachvollziehbaren Claude-Code-Workflow etablieren

06 FAQ

Häufige Fragen

Ist der Kurs eine allgemeine Claude-Schulung?

Nein. Im Mittelpunkt steht Claude Code als Werkzeug für Softwareentwicklung und Engineering. Allgemeines Prompt Engineering wird nur dort behandelt, wo es für die Arbeit mit Claude Code relevant ist.

Muss ich Claude Code bereits verwendet haben?

Nein. Der Kurs beginnt mit Einrichtung und grundlegender Arbeitsweise und führt anschließend zu Projektinstruktionen, Hooks, MCP, Subagents und komplexeren agentischen Workflows.

Wird CLAUDE.md behandelt?

Ja. Projektwissen und dauerhafte Instruktionen sind ein zentraler Bestandteil des Kurses. Dabei geht es insbesondere darum, welche Informationen tatsächlich hilfreich sind und wie Projektinstruktionen wartbar bleiben.

Werden Hooks behandelt?

Ja. Hooks werden genutzt, um wiederkehrende Aktionen und Kontrollen technisch in den Workflow einzubinden.

Wird MCP behandelt?

Ja. Claude Code kann über MCP zusätzliche Werkzeuge und Datenquellen einbinden. Im Kurs werden sowohl die Nutzung vorhandener MCP-Server als auch die Gestaltung eigener Tool-Anbindungen behandelt.

Werden Subagents behandelt?

Ja. Spezialisierte Agenten werden für abgegrenzte Analyse-, Entwicklungs- und Review-Aufgaben eingesetzt. Dabei wird auch betrachtet, wann zusätzliche Agenten tatsächlich einen Vorteil bringen.

Wird mit der Shell gearbeitet?

Ja. Die Shell ist ein wesentlicher Bestandteil von Claude Code. Gerade deshalb werden Berechtigungen, Auswirkungen von Kommandos und die kontrollierte Nutzung von Entwicklungswerkzeugen ausführlich behandelt.

Geht es auch um Sicherheit?

Ja. Bei einem agentischen Werkzeug reicht es nicht, nur über gute Prompts zu sprechen. Berechtigungen, Secrets, externe Systeme, Tool-Zugriffe, reversible Aktionen und technische Kontrollmechanismen gehören deshalb ausdrücklich zum Kurs.

Werden eigene MCP-Tools behandelt?

Ja. Der Kurs behandelt nicht nur die Konfiguration vorhandener MCP-Server, sondern auch die grundlegende Gestaltung sinnvoller eigener Tool-Schnittstellen.

Warum dauert der Kurs drei Tage?

Claude Code umfasst deutlich mehr als Codegenerierung. Repository-Arbeit, Projektinstruktionen, Shell und Tools, Tests, Hooks, MCP, Subagents, Berechtigungen und vollständige agentische Entwicklungsabläufe müssen praktisch erarbeitet werden. Drei Tage sind dafür eine realistische Untergrenze, ohne den Kurs zu einer reinen Funktionsdemonstration zu machen.

Kann der Kurs an unsere Entwicklungsumgebung angepasst werden?

Ja. Bei Inhouse-Schulungen können vorhandene Repositories, Programmiersprachen, Build- und Testwerkzeuge, CI/CD, MCP-Server und interne Entwicklungswerkzeuge in die Übungen einbezogen werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/claude-code/