Claude Code

基礎から習得まで

2026年版 エージェンティックな
ソフトウェア開発の完全ガイド

- インストールとはじめ方
- プロンプトエンジニアリング
- マルチエージェントワークフロー
- セキュリティとベストプラクティス

- 実践的なプロジェクト
- Stripe・Wizの事例研究
- CI/CD自動化
- エージェンティックコーディングの未来

Claude Codeを使いこなし、ソフトウェア開発のつくり方を変革しましょう。
初期セットアップから本番環境への展開まで——信頼できるエンジニアリング
チームが実践するツール・手法・ワークフローを網羅的に解説します。

- 各章に実用的な具体例を掲載
- すぐに役立つハンズオンプロジェクト
- 現場のチームから学ぶ実践知
- 10万回以上のセッションで検証された内容

Steve T.

Claude Code：基礎から習得まで

2026 年版：エージェンティック・ソフトウェア開発の完全ガイド

Steve T. Publications

本書はこちらで販売中です https://leanpub.com/claudecodejp

この版は 2026-07-15 に発行されました。

本書は Leanpub の電子書籍です。Leanpub はリーンパブリッシングプロセスで著者や出版社を支援します。リーンパブリッシング は新しい出版スタイルです。軽量なツールを使って執筆中の電子書籍を出版し、読者のフィードバックをもらいながら魅力的な本に仕上がるまでピボットを繰り返すことができます。

Contents

2026 年エージェント型ソフトウェア開発完全ガイド

Claude Code、マルチエージェントワークフロー、およびエンタープライズ導入をマスターするための包括的なハンドブック

第 1 版 2026 年 7 月

本書は Claude Code version 2.1.131 および Claude モデルの Opus 4.8、Fable 5、Sonnet 4.6、Haiku 4.5 までをカバーしています。すべての価格、ベンチマーク、機能説明は 2026 年 7 月時点のプラットフォームの状態を反映しています。

本書について: これは Anthropic のエージェント型コーディングシステムである Claude Code の包括的なガイドです。このシステムはソフトウェアの開発方法を根本的に変えました。AI 支援プログラミングに初めて取り組む開発者であれ、マルチエージェントワークフローをマスターしようとする経験豊富なエンジニアであれ、本書は最初のインストールからエンタープライズ導入までをサポートします。Claude Code のアーキテクチャ、プロンプトエンジニアリング技法、Stripe や Wiz の実践事例、CI/CD 自動化、セキュリティベストプラクティス、そしてエージェント型コーディングの未来をカバーしています。各章には実践的な例、ハンズオンプロジェクト、数万に及ぶ実際の Claude Code セッションの研究に基づく実行可能なガイダンスが含まれています。

はじめに：もはやタイプしないエンジニア

2025 年後半、Anthropic の Claude Code の創作者であり責任者である Boris Cherny は、わずか 3 年前であれば SF のように聞こえたであろう観察をしました。彼は前年の 11 月以来、手動で 1 行のコードも書いていなかったのです。その代わりに、彼は望むものを説明し、Claude Code がどのように構築するかを解決し、テストを実行し、失敗を修正し、プルリクエストを送信しました。彼は 5 つのローカルターミナルセッションと 5〜10 のリモートインスタンスを同時に実行し、毎日 10〜30 のプルリクエストを送信していました [1]。

ツールの創作者が、そのツールを構築した会社が実際に毎日どのように運用しているかを説明しており、これはより大きな物語におけるただの 1 つのデータポイントに過ぎません。

2026 年 5 月現在、Anthropic 自身の本番コードベースにマージされたコードの 80% 以上が Claude によって作成されており、2025 年 2 月に研究プレビューとして Claude Code がローンチされる前の低個桁率から大幅に上昇しています [7]。Anthropic のエンジニアはゼロから実装をタイプしません。彼らは望むものを説明し、Claude Code がどのように構築するかを解決し、テストを実行し、失敗を修正し、プルリクエストを送信します。人間のエンジニアは出力をレビューし、アーキテクチャ上の決定を行い、次の問題に進みます [8]。

本書で学ぶこと

本書は Anthropic のエージェント型コーディングシステムである Claude Code の包括的なガイドです。このシステムはソフトウェアの開発方法を根本的に変えました。AI 支援プログラミングに初めて取り組む開発者であれ、マルチエージェントワークフローをマスターしようとする経験豊富なエンジニアであれ、本書は最初のインストールからエンタープライズ導入までをサポートします。Claude Code のアーキテクチャ、プロンプトエンジニアリング技法、Stripe、Wiz、Rakuten の実践事例、CI/CD 自動化、セキュリティベストプラクティス、そしてエージェント型コーディングの未来をカバーしています。各章には実践的な例、ハンズオンプロジェクト、数万に及ぶ実際の Claude Code セッションの研究に基づく実行可能なガイダンスが含まれています。

本書を終了するまでに、あなたは以下ができるようになります：

- 複数のプラットフォームおよび IDE 間で Claude Code をインストールし、設定する

- 最初の試行で確実に本番品質のコードを生成するプロンプトを書く
- インタビューパターンと仕様駆動開発を使用して、ゼロからフルスタックアプリケーションを構築する
- Claude があなたの介入なしに自己修正できるように、テストおよび検証ループを設定する
- ヘッドレスモード、ファンアウトパターン、スケジュール済みルーチンを使用して、反復的なタスクを自動化する
- リーダージェント、専門サブエージェント、共有ファイルシステムを持つマルチエージェントアーキテクチャを設計する
- マネージド設定、SSO、ガバナンスポリシーを使用して、エンタープライズスケールで Claude Code を導入する
- プロンプトインジェクション、シークレットの漏洩、サプライチェーン攻撃を含め、エージェント型コーディングツールに固有のセキュリティリスクを理解し、軽減する

本書の対象読者

本書はソフトウェア開発についての基本的な知識を前提としています：ターミナルが何か知っている、少なくとも 1 つの言語でコードを書いたことがある、バージョン管理、テスト、デプロイメントのような概念を理解しています。特定のプログラミング言語の専門家である必要はありません。Claude Code は言語を横断して動作し、本書で教えるメンタルモデルはあなたの技術スタックに関係なく移行可能です。

自動補完ツールを使用して、役には立つが限定的だと感じたジュニア開発者であれば、本書は自動補完を超えた先に何が待ち受けているかを示します。あなたの組織のために Claude Code を評価しているシニアエンジニアまたはチームリードであれば、本書は情報に基づいた決定を下すために必要な技術的な深さと戦略的文脈を提供します。プロダクトマネージャー、デザイナー、創業者、コードを 1 行ずつ書かずにソフトウェアを構築したいノンテクニカルなプロフェッショナルであれば、本書は Claude Code がどのようにその扉を開くかを示します。

本書の使い方

本書は 3 つのパートに構成されており、それぞれが前のパートの上に構築されています：

パート I（第 1〜3 章）：基盤。第 1 章と第 2 章は概念的な枠組みを確立します。Claude Code が AI コーディングツールの進化の中でどこに位置するか、そのエージェント型アーキテクチャが自動補完やインラインアシスタントとどのように異なるか、内部ループがど

のように動作するかを理解します。第 3 章はインストールと最初のセッションを案内します。

パート II（第 4〜8 章）：コアスキル。第 4 章から第 6 章は Claude Code との実践的なスキルの作業を教えます：プロンプトエンジニアリング、実践的な開発ワークフロー、テスト戦略。第 7 章と第 8 章は自動化、スケーリング、高度なエージェントパターンをカバーします。ここで行うのは、生産的なセッションとフラストレーションの溜まるセッションを分ける筋肉記憶を発達させることです。

パート III（第 9〜13 章）：本番運用以降。第 9 章と第 10 章はデプロイメント、可観測性、エンタープライズコラボレーションを取り上げます。第 11 章はセキュリティのディープダイブです。第 12 章はエージェント型コーディングの未来を展望します。第 13 章はすべてを実行可能なガイダンスに統合し、効果的なエージェント型エンジニアになる方法を導きます。

本書を最初から最後まで読むことも、現在のニーズに最も関連する章にジャンプすることもできます。各章は事前に紹介された概念の上に構築しながら、独立したディープダイブとして設計されています。

証拠ベース

本書の証拠ベースには、公式 Anthropic ドキュメントおよびエンジニアリングブログ記事の広範な分析、2025 年 10 月から 2026 年 4 月の間に約 235,000 人のユーザーから約 400,000 の Claude Code セッションに関する同社の研究 [6]、claude.com/customers/ に Anthropic が公開した顧客ケーススタディ、Check Point Research および他のセキュリティ企業からの第三者技術分析、コミュニティが貢献したワークフローが含まれています。主張が独立して検証できない場合は、その情報源に帰属されます。証拠が不十分または争われている場合は、本文でそれらが言及されています。

本書がカバーしないことに関する注記

本書は、自分のコードベースと作業するための開発ツールとしての Claude Code に焦点を当てています。Claude API や Claude API を使用してアプリケーションを構築することはカバーしていませんが、本書の概念は Claude の基礎モデルがどのように動作するかを理解する強い基盤を提供します。また、Anthropic のポートフォリオ内の別のプロダクトである Claude Cowork（コラボレーティブワークスペース）、Claude Design、Claude Security もカバーしていません。

中心的なテーゼ

本書の中心的なテーゼは端的です：Claude Code はコード補完からエージェント型コーディングへのパラダイムシフトを表しています。人間は目標を定義し、成果をレビューするオーケストレーターとなり、Claude が実行を担当します。しかし、このパラダイムシフトには現実的なリスクが伴います：設定ファイルのセキュリティ脆弱性、コンテキスト管理の課題、品質低下インシデント、エンタープライズグレードのガバナンスの必要性。このツールをマスターするには、コマンドを学ぶだけでなく、委任、検証、自律エージェントとのコラボレーションのための新しいメンタルモデルを開発する必要があります。

2026 年の状況

Claude Code は利用可能なエージェント型コーディングツールの唯一ではありません。競争環境には GitHub Copilot、Cursor、OpenAI Codex、Google Antigravity、その他が含まれ、それぞれが独自の戦略と強みを持っています。それらの違いを理解することは、あなたのワークフローに適したツールを選ぶために不可欠です。しかし Claude Code は独特な位置を占めています：ターミナルファーストですが複数のインターフェースで動作し、ファイルレベルではなくプロジェクトレベルで動作し、エンタープライズチームの中で最も積極的な採用を示しています。

この技術は急速に進んでいます。Claude Code はローンチ以来大きな変化を経ており、2026 年 5 月 6 日の「Code with Claude」カンファレンスでは、Dreaming（セッション間メモリキュレーション）、Outcomes（自動化品質評価）、マルチエージェントオーケストレーションを追加する主要な機能リリースが行われました [3]。モデルラインナップには、SWE-bench Verified で 95.0% を達成する Claude Fable 5、88.6% の Claude Opus 4.8、本番デフォルトの Claude Sonnet 4.6 が含まれています [2]。効果的な使用のコア原則、委任と検証のメンタルモデル、Claude Code がどのように動作するかというアーキテクチャ的理解は、モデルアップデートに関係なく不変です。

以下の章は Claude Code をマスターするための完全なガイドを提供します。

第 1 章：ソフトウェア開発におけるエージェント型革命

自動補完からエージェントへ：5 年間の進化

Claude Code がこの状況の中でどこに位置するかを理解するには、AI 支援コーディングの約 5 年間での進化を追うことが役立ちます。この進展は 4 つの明確なフェーズを経過しており、それぞれが前のフェーズの上に構築され、可能となる範囲を拡大してきました。

最初のフェーズ（約 2019 年から 2021 年）は、ルールベースのコード補完でした。TabNine のようなツールはオープンソースコードで訓練された統計モデルを使用して、開発者がタイプする際に次の行や関数を提案しました。これらのツールは定型文や一般的なパターンにとって確かに役に立ちましたが、意図、アーキテクチャ、複数ファイルの変更について推論することはできませんでした。それらは行を補完しましたが、プロジェクトを理解することはできませんでした。

2 番目のフェーズ（2021 年から約 2024 年半ば）は、大規模言語モデル（LLM）自動補完でした。2021 年に OpenAI の Codex モデルとともにローンチされた GitHub Copilot は量子飛躍を表しました。統計的予測の代わりに、ニューラル言語モデルを使用してコード提案を生成しました。開発者は関数を選択してテストスイートを要求したり、コメントで望むものを説明して動作する実装を受け取ったりできました。この体験は多くの開発者にとって変革的でしたが、本質的には受動的なままでした。開発者がタイプし、モデルが応答し、開発者が各提案を受け入れるか拒否しました。

3 番目のフェーズ（2024 年後半に登場し、2025 年を通じて加速）は、インラインアシスタントと AI ネイティブ IDE を導入しました。Cursor（VS Code のフォーク）や Claude 自身のインテグレーションのようなツールは、単一ファイルの提案を超えました。それらはプロジェクト全体のコンテキストを読み、ファイルが互いにどのように関連しているか理解し、複数のファイルにわたって調整された変更を行うことができました。開発者はコードベースについて質問し、実際のコード構造に基づいた答えを受け取ることができました。

2025 年 5 月 22 日に Claude 4 とともに一般利用可能なツールとしてローンチされた Claude Code は、第 4 のフェーズ：エージェント型コーディングを表しています。先行者与異なり、Claude Code は提案が受け入れられたり拒否されたりするのを待ちません。目標を受け取り、一連のアクションを計画し、実際の開発ツール（ファイルの読み込み、コードの編集、テストの実行、git の使用）を使用してそれらを実行し、結果を評価し、アプローチ

を調整します。開発者は各ステップをガイドするのではなく、目的を定義し、結果をレビューします [18]。

自動補完からエージェントへの移行は、人間と機械の基本的な関係を変えます。自動補完では、あなたが作業を行い、ツールが提案で補助します。エージェント型システムでは、計画して、実行して、独立して反復する有能なコラボレーターに作業を委任します。

Anthropic 自身のデータはこの進化を反映しています。2025 年 10 月から 2026 年 4 月の約 400,000 の Claude Code セッションに対するプライバシー保護分析は、その 7 ヶ月間で作業の構成が劇的に変化☒ revealed [6]。壊れたコードの修正に費やされたセッションの割合は 33% から 19% に減少し、ソフトウェアの運用は 14% から 21% に増加しました。新しいコードの記述とデータ分析は約倍増し、約 10% から 20% になりました。タスク自体もより価値あるものとなりました：平均セッションの推定経済的価値は 27% 上昇しました。

エージェントがより有能になるにつれて、開発者は小さな修正に使用するのをやめ、エンドツーエンドの開発、デプロイメント、運用に使い始めます。ツールはもはやコードを書くだけでなく、パイプラインを実行し、アプリケーションをデプロイし、データを分析し、ドキュメントを生成しています。

Claude Code を異なるものにするもの：ターミナルファーストのエージェント型アーキテクチャ

2026 年の AI コーディングツールの状況は、それぞれが独自の戦略と強みのセットを持つ数つの主要プラットフォームを中心に統合されました。それらの違いを理解することは、あなたのワークフローに適したツールを選ぶために不可欠です。

GitHub Copilot は Microsoft および GitHub によって保守されており、最も広範に互換性のあるオプションです。VS Code、JetBrains、Neovim、Xcode、Eclipse、Zed、SQL Server Management Studio を含む 10 以上の IDE と統合されます。その強みは配布とエコシステム統合にあります。Copilot は OpenAI（GPT-5.1〜5.4）、Anthropic（Haiku 4.5、Sonnet 4.6、Opus 4.6）、Google（Gemini 3 Pro および Flash）、xAI Grok Code にわたるモデルの柔軟性をサポートします。エンタープライズティアはカスタムナレッジベース、プルリクエストサマリー、細粒度の管理者コントロール、SAML シングルサインオンを追加します。個人開発者で月額 $10、Business ティアチームでシートあたり月額 $19 で、基本的な AI コーディング支援にとって最高のコストパフォーマンスオプションであり続けます。その弱みは GitHub エコシステムとの密結合と、自律的なマルチステップ実行能力の制限です。

Cursor は Anysphere によって開発され、VS Code フォーク上に構築された完全な AI ネイティブ IDE です。その強みは統合の深さにあります。すべての編集レイヤーは AI 対応です：タブ補完、複数ファイルのエージェント型編集用の Composer 2、バックグラウンドク

ラウドエージェント、自動化プルリクエストレビュー用の Bugbot。Cursor は個人開発者で月額 $20、チームでシートあたり月額 $40 で、最も深い AI ネイティブ IDE 体験を提供します。SOC 2 Type 2 認証取得、年次ペネトレーションテスト、SCIM シート管理、SAML/OIDC シングルサインオンを備えています。主な制限は、フルチームの Cursor IDE への移行が必要であり、無料 Hobby ティアはクラウドエージェントと高度な機能を除外していることです。

Claude Code は Anthropic によって開発され、根本的に異なるアプローチを取ります。ターミナルファーストですが複数のインターフェースで動作します：CLI、VS Code 拡張機能、JetBrains プラグイン、デスクトップアプリ、claude.ai/code の Web インターフェース、Slack、CI/CD パイプライン。その定義特性はエージェント型能力です。Claude Code はプロジェクトレベルで動作し、フルコードベースを読み、複数のファイルにわたって計画し、変更を実行し、テストを実行し、失敗に対して反復します。開発者は目標を定義し、結果をレビューします [18]。

2026 年初頭の競争環境データは、開発者の好みに際立つ変化を示しています。Pragmatic Engineer の 2026 年 2 月の 15,000 人のソフトウェアエンジニアに対する調査で、Claude Code は 46% の「最も愛されている」評価を達成し、Cursor は 19%、GitHub Copilot は 9% でした [50]。これは機能だけの問題ではありません。エージェント型能力が自動補完よりも重要であるという広まる認識を反映しています。

以下の比較表は主要な違いを要約します：

次元	Claude Code	GitHub Copilot	Cursor
プライマリーインターフェース	ターミナル + IDE 拡張機能	IDE プラグイン（10 以上のエディター）	AI ネイティブ IDE（VS Code フォーク）
エージェント型能力	完全なマルチステップ自律性	インライン提案 + ワークスペース	複数ファイル編集、クラウドエージェント
モデルオプション	Anthropic モデルのみ	OpenAI、Anthropic、Google、xAI	複数のモデルプロバイダー
外部ツール統合	MCP サーバー（オープンスタンダード）	GitHub エコシステム拡張機能	Cursor 専用拡張機能
Slack 統合	ネイティブ非同期コーディングワークフロー	限定的	なし

次元	Claude Code	GitHub Copilot	Cursor
エンタープライズセキュリティ	HIPAA 対応、SSO、RBAC	SAML SSO、IP 免責	SOC 2 Type 2、ゼロデータ保持
個人価格	$20/月（Pro）	$10/月（Pro）	$20/月（Pro）

これらのツールの選択は必ずしも二者択一ではありません。多くの大規模エンジニアチームは、Copilot をマルチ IDE の広範囲のために実行しつつ、深いコードベース推論と自律的実行が必要な高レバレッジな複雑タスクのために Claude Code や Cursor を導入しています。重要な要因はツールをタスクにマッチさせることです：単純な自動補完は広い IDE 互換性から恩恵を受け、複雑な複数ファイルのリファクタリングとエンドツーエンド機能開発は Claude Code のエージェント型アーキテクチャから恩恵を受けます。

Boris Cherny のワークフロー：創作者が実際に Claude Code を使用する方法

Claude Code をどのように使用するべきかについての最良の証拠は、その創作者から来ます。以前 Meta で会社全体のコード品質を担当していた Boris Cherny は、Anthropic で Claude Code チームを率いています。コード品質エンジニアリングから AI コーディングツール構築への彼のキャリア軌跡は、開発者が何が必要かについて独特な視点を与えてくれます [49]。2025 年後半以来、彼のコードの 100% が Claude Code によって書かれています。彼は毎日 10〜30 のプルリクエストを送信し、5 つのローカルターミナルセッションと並行して 5〜10 のリモートインスタンスを実行しています [1]。

Cherny のワークフローは体系的であり、カジュアルではありません。カスタマイズを避け、Claude Code をそのまま使用します。各ローカルセッションはブランチやワークツリーの代わりに個別の`git checkout` を使用し、並列セッション間の競合を防ぎます。リモートセッションは& 演算子によってバックグラウンド化され、`--teleport` でシフトされますが、予期せぬシナリオのために 10〜20% が放棄されると述べています [2,29]。

コーディングにおいて、Cherny は thinking を有効にした Opus 4.5（現在は Opus 4.8 に後継）を好み、Sonnet と比較して優れた品質、信頼性、ツール使用能力を挙げています。彼は Sonnet がステップあたり速いかもしれませんが、最初の試行がより頻繁に成功するため、全体的なスループットは Opus の方が速いと述べています [2,29,47]。

知識保持は git 内のチーム管理の`CLAUDE.md` ファイルによって制度化されています。このドキュメントは間違い、スタイル規約、設計ガイドライン、PR テンプレートをログに記録し、AI を反復的に改善します。Cherny は同僚のプルリクエストに`@.claude` タグを付けて新しい学習を注入し、ファイルを約 2.5k トークンに保っています [2]。これは重要な洞察で

す：CLAUDE.md は静的な設定ファイルではなく、集合的なチーム経験を通じて時間とともに改善される生きているドキュメントです。

彼のプロセスは Plan モードを使用した反復的計画フェーズから始まり、その後自動受け取り編集に切り替わり、通常最初の試行で成功します。コミットや PR のような日常運用は.claude/commands/に保存されたスラッシュコマンドによって自動化され、明示的なプロンプトを最小限に抑えます。/commit-push-pr コマンドはモデルレイテンシを削減するために git ステータスをインラインで事前計算することで、毎日何十回も実行されます。フォーマットの不整合からの CI 失敗を防ぐために、彼は WriteEdit マッチに対して自動的にbun run format || true を実行するPostToolUse フックを設定しています [2,46]。

セキュリティは安全な bash コマンド（例：bun run build:*、cc:*）を/permissions を通じて付与することで処理され、--dangerously-skip-permissions の必要性を大幅に削減します。危険なフラグはサンドボックス化された長時間実行タスクに専ら予約され、反復的な中断を防ぎます。Cherny は検証フィードバックループ（テストスイートの実行やブラウザ自動化など）を提供することが重要であると強調し、最終出力品質を 2〜3 倍に向上させます。彼は claude.ai/code にラウンドされたすべての変更を Claude Chrome 拡張機能を使用してテストしていると述べています [2,48]。

この構造化されたアプローチは、コードがすでに磨かれた状態でレビューに到着することを保証し、エンジニアが誘導とアーキテクチャの監督に集中できるようにします。パターンは明確です：まず計画し、次にルーチンステップを自動化し、厳格に検証し、Claude に実行を任せます。

労働の分担：オーケストレーターとしての人間、ビルダーとしてのエージェント

Anthropic の 400,000 の Claude Code セッションに関する研究からの最も重要な発見の 1 つは、人間とエージェントの間で生じる明確な労働の分担です [6]。典型的なセッションでは、人々は計画決定（何をすべきか、どのアプローチを取るべきか、何が完了とカウントされるか）の約 70% を行い、Claude は実行決定（どのファイルを変更すべきか、什么コードを書くべきか、どのコマンドを実行すべきか）の約 80% を行います。

この分担は偶然ではありません。人間の好みとエージェントの能力の両方を反映しています。人間は自然に計画と意思決定に引き寄せられます。なぜなら、それがドメインエキスパートが最も重要な場所だからです。Claude に調整スクリプトを構築させるよう指示する会計士は、ビジネスルール、月末締め時のエッジケース、成功を定義する検証基準の知識をもたらします。Claude はコード構造、ライブラリ API、テストフレームワーク、デバッグ技法の知識をもたらします。

データは職業間で一貫してこの分担を示しています。コンピュータおよび数学的職業（最大のユーザーグループ）は全体的に約 30% のセッションで検証済み成功に達しています [6]。他の職業からのユーザーは約 26% の頻度で検証済み成功に達します。コードを生成するセッションの中では、それらの数値はそれぞれ 34% および 29% です。10 つの最大非ソフトウェア職業グループのすべてが、成功率においてソフトウェアエンジニアから 7 ポイント以内に収まります。管理職業は実際には検証済み成功で最高ランクであり、ソフトウェアエンジニアリング職業をわずかに上回ります。

人がセッションにもたらすドメインエキスパートが大きいほど、Claude は指示あたりに多くの作業を行います。典型的な初心者セッションでは、各プロンプトは約 5 つの Claude アクションと約 600 語の出力を起動します。エキスパートセッションは 2 倍以上長いアクションチェーン（12 アクション）と 5 倍の出力（3,200 語）を起動します [6]。このギャップはあらゆる種類の作業とタスク価値のすべてのバンドに現れます。

コラボレーションの形状は Claude Code を考える方法にとって重要です。それは開発者の代替ではありません。それはドメインエキスパートのフォースマルチプライヤーです。最初の Rust 質問をするシニアエンジニアは Rust の初心者です。Python を使用したことがないが、Claude に Python スクリプトがどの調整ルールを強制すべきかを正確に伝え、月末締め時に誤処理するエッジケースを捉える会計士はそのタスクのエキスパートです [6]。このツールはプログラミング言語の習熟度ではなく、問題ドメインの理解を報います。

訓練は新しい構文や API リファレンスの学習ではなく、問題定義の明確さ、検証基準、ドメイン知識の開発に焦点を当てるべきです。最も効果的な Claude Code ユーザーは必ずしも最高のコーダーではありません。彼らは何を構築すべきか、なぜそれが重要なのか、どのように動作するかを検証するかを説明できる人々です。

2026 年の AI コーディングエージェントの状況

Claude Code は利用可能なエージェント型コーディングツールの唯一ではありません。競争環境にはいくつかの注目すべきプレイヤーが含まれています：

OpenAI Codex は、初期 GitHub Copilot 機能を駆動する元の Codex モデルの後継者であり、スタンドアロンのエージェント型システムに進化しました。複数ファイル編集と自律的実行で Claude Code と同様に動作します。OpenAI の強みは膨大な開発者ベースと ChatGPT Workspace との統合にあります。弱みはエンタープライズ使用のためのより不透明な価格モデルです。

Google Antigravity 2.0 は 2026 年初頭に発表され、コーディングタスクのために Google の Gemini モデルを活用しています。Google Cloud および Vertex AI と深く統合され、すでに Google エコシステムに投資しているチームにとって自然な選択になります。

SWE-bench でのベンチマークパフォーマンスは Claude Fable 5 および OpenAI Codex に遅れています [8]。

Devin は Cognition Labs（Amazon によって買収）の元の自律的コーディングエージェントであり、フルソフトウェアタスクを計画して実行できるエージェントの概念を先導しました。カテゴリを確立しましたが、2026 年半ば時点でほとんどのベンチマークで Claude Code に能力を超えられました。

Amazon Q Developer は AWS エコシステムに組み込まれ、AWS サービスとの深い統合を持つエージェント型コーディング機能を提供します。インフラストラクチャが強く AWS 依存のチームにとって強力な選択です。

この空間の競争動態は急速に変化しています。かつてモデルレース（どの LLM がコーディングで最も賢いか）だったものが、ハーネスレース（どのプラットフォームがモデル能力、外部ツール、人間の監督を最もよくオーケストレートするか）になりました。Anthropic の 2026 年 5 月「Code with Claude」イベントでの戦略はこれを強化しました：新しいモデルは発表されませんでした。代わりに、5 つのインフラストラクチャ機能がリリースされました：Dreaming、Outcomes、マルチエージェントオーケストレーション、10 つの事前に構築されたエージェントを持つ Claude Finance、Add-ins [3,38]。メッセージは明確でした。本番エージェントシステムのボトルネックはモデル能力ではありません。それはモデルを取り巻く足場です。

なぜ今なのか？市場採用とエージェント型ワークフローへの移行

2026 年におけるエージェント型コーディング採用の加速は、いくつかの収束する要因によって駆動されています。

第一に、モデル能力は自律的エージェントが信頼性の高いマルチステップタスクを処理できる閾値に達しました。Claude Fable 5 は SWE-bench Verified で 95% を達成 [22] し、Claude Opus 4.8（2026 年 5 月 28 日リリース）は前世代よりも約 4 倍少ない見逃されたコード欠陥で、著しく優れたエージェント型判断を提供します [28]。これらは信頼性における質的なシフトを表しています。

第二に、大規模にエージェントを実行するためのインフラストラクチャが成熟しました。プロンプトキャッシングは Claude Code セッション間で 92% のプレフィックス再利用率を達成し、コストを約 81% 削減し、継続的なエージェント使用を経済的に可能にしています [26]。Agentic AI Foundation によって管理されている Model Context Protocol（MCP）は、Anthropic、OpenAI Agents SDK、Cursor、Cloudflare Agents、Microsoft によって実装されており [17]、エージェントからツールへの統合のためのオープンスタンダードを作成しています。

第三に、市場採用はティッピングポイントを越えました。コーディングエージェントアクティビティを持つ GitHub プロジェクトの割合は 2025 年後半以来 2 倍以上になっています [6]。Claude Code は 2026 年 2 月に年間 $25 億の収益を越えました [8]。Anthropic 自身では、80% の新しい本番コードが現在 Claude Code によって作成されています [7]。コミュニティの勢いはエンタープライズ採用と並行して進行しています：GitHub 上の Claude Code リポジトリは 2026 年半ばまでに 127,000 スターを超えました [25]。

第四に、経済的インセンティブは説得力があります。Stripe、Wiz、Ramp、Rakuten からのケーススタディは、無視できない生産性向上を示しています。50,000 行の Python ライブラリを 2〜3 ヶ月の代わりに 20 時間で Go に移行できるチームは、その容量制約を根本的に変えました。機能配信を 24 日から 5 日に削減する会社は、プロダクト開発サイクルを約 80% 圧縮しました。

エージェント型コーディングへの移行は一時的なトレンドやハイプサイクルではありません。それはソフトウェアがどのように構築されるかの構造的変化を表しており、パンチカードからキーボードへ、またはアセンブリ言語から高級言語への移行に匹敵します。ツールは変わっており、ワークフローは変わっており、最も重要なスキルは変わっています。

問題はもはやエージェント型コーディングがメインストリームになるかどうかではなく、チームがどのようにそれに対して効果的に適応できるかです。本書はその質問に答えるために必要な技術的な深さと戦略的文脈を提供することを目指しています。

批判的分析：エージェント型ツールのトレードオフと制限

Claude Code を効果的に使用する前に、エージェント型コーディングツールができないことと、従来の開発ワークフローを超えてどこで新しいリスクを導入するかを検証する価値があります。これらの制限を理解することは、コストのかかる間違いを防ぎます。

コンテキストローテーションは現実です。 1M トークンウィンドウがあっても、広範なツール出力、エラーログ、複数ファイルの diff を蓄積するセッションは最終的に自動圧縮をトリガーします。圧縮が発生すると、Claude は以前の指示やセッションの前半に確立された微妙なコンテキストを追跡しなくなる可能性があります。Anthropic の 2026 年 4 月の品質低下インシデントはこれを鮮明に示しました：すべてのターンで推論履歴をクリアするキャッシングバグが、何千ものセッションで物忘れ、反復、奇妙なツール選択を引き起こしました [12]。2026 年 4 月 23 日のポストモーテムは、3 つの個別のプロダクトレイヤー変更が 2026 年 3 月 4 日から 4 月 20 日の間に低下を引き起こしたことを明らかにしました：high から medium へのデフォルト推論努力のシフト（3 月 4 日）、アイドル時間あたり 1 回ではなくすべてのターンで推論を破棄するキャッシング最適化バグ（3 月 26 日）、ツール呼び出しテキストを 25 語未満に制限するシステムプロンプト冗長性削減（4 月 16 日）

[12,33,34]。すべては version 2.1.116 の 4 月 20 までに修正されました。教訓は Claude Code が信頼できないというのではなく、LLM を取り巻くハーネスのわずか 1 つの微妙な変更で、大規模なユーザーベースにわたって測定可能な品質低下につながる可能性があるということです。

Claude Code はテストされていないタスクを信頼性を持って処理できません。 SWE-bench はオープンソースリポジトリ内の既知のバグを修正するモデルの能力を測定します。それは馴染みのないアプリケーションをゼロから構築するモデルの能力を測定しません。Claude Code は両方できますが、信頼レベルは著しく異なります。Fable 5 の 95% SWE-bench Verified スコアは、既知のテストケースと明確に定義された失敗シグナルを持つ特定のベンチマークでのパフォーマンスを反映しています。高レベルの説明から新しいアプリケーションを構築することは、曖昧な要件、アーキテクチャ決定、どのベンチマークもキャプチャしないトレードオフを伴います。

ベンチマークパフォーマンスと現実の世界の能力の間のギャップを理解するには、SWE-bench がどのように動作するかを理解する必要があります。各 SWE-bench タスクは実際のオープンソースプロジェクトからの GitHub issue であり、issue が提起された時点のリポジトリ状態と修正が正しいかを定義する既存のテストセットとペアになっています。モデルは issue 説明を受け取り、リポジトリを探索し、既存のテストをパスさせるコード変更を書くことを試みます。ベンチマークには現実の開発よりも簡単にする 3 つの特性があります：問題は明確に定義されている（issue は特定のバグを説明する）、成功基準は決定論的である（テストはパスするか失敗するか）、コードベースは馴染み深い（モデルはこれらのリポジトリを含むオープンソースコードで訓練されています）。

現実の開発はこの 3 つの特性のすべてを欠いています。Claude に「タスク管理 API を構築して」と依頼すると、要件は曖昧です（タスクは何らのフィールドを持つべきか？どのような認証モデル？）、成功基準は主観的です（API は適切に感じられるか？エラーハンドリングは十分か？）、コードベースは完全に新しい可能性があります（Claude はあなたのプロジェクトを以前に見たことがありません）。95% の SWE-bench スコアは、Claude が現実の開発タスクで 95% の頻度で成功することを意味しません。それは Claude が決定論的テストスイートを持つ明確に定義されたバグ修正タスクの特定のクラスで 95% の頻度で成功することを意味します。

この区別はアーキテクチャ決定にとって重要です。包括的なテストスイートを持つ既存のコードベースでバグを修正するために Claude Code を使用している場合、出力に対して高い信頼を持てます。ゼロから新しいシステムを設計して実装するために Claude Code を使用している場合、著しく多くの人間の監督、検証、反復的洗練が必要です。ベンチマークスコアは生のコーディング能力の有効なシグナルですが、新しいタスクでの現実の世界の成功の信頼できる予測因子ではありません。

Claude Code はマルチステップチェーンで幻覚を起こす可能性があります。単一のステップは正しい出力を生成するかもしれませんが、エラーは 10〜20 のステップのチェー

ンで累積します。ステップ 3 で読まれるべきだったファイルがステップ 5 によって編集されており、Claude のコードベース状態の理解は古くなります。これが各ステップでの検証が重要な理由です。Huang ら（2026）による研究は、Claude Code を含む 7 つの広く使用されている MCP クライアントをテストし、ツールポイズニング攻撃が接続された外部システムから返されたデータを通じて悪意のある命令を注入できることを発見しました [14]。Claude が組み込みの命令を含む GitHub README、データベースクエリ結果、または Slack メッセージを読むとき、それらの命令はコンテキストの一部になり、その動作を操作できます。

Claude Code はビジネス要件について決定できません。要求されたものを実装できますが、機能が存在すべきか、どの API 設計が最適か、どのトレードオフが許容可能かを決定することはできません。これらは人間の判断です。400,000 セッションに関する研究はこれを裏付けています：人間は約 70% の計画決定を行い、Claude は約 80% の実行決定を処理します [6]。

Claude Code は開発環境に現実的なセキュリティリスクを導入します。 Check Point Research によって記録されているように、CVE-2025-59536 は信頼ダイアログが表示される前に悪意のあるプロジェクト設定ファイルを通じてリモートコード実行を可能にし、CVE-2026-21852 は ANTHROPIC_BASE_URL 操作による API キー窃取を可能にしました [11]。これらは実際のデプロイメントに影響を与え、従来パッシブメタデータとして扱われていた設定ファイルが実行可能な攻撃ベクトルになることを示した文書化された脆弱性です。

Claude Code を超えて、より広い MCP エコシステムは体系的なセキュリティ課題に直面しています。Ox Security からの脆弱性アドバイザリ（2026 年 4 月）は、Windsurf、Claude Code、Cursor、Gemini CLI、GitHub Copilot を含む IDE およびコーディングアシスタントが、MCP JSON 設定ファイル経由のコマンドインジェクションに対して脆弱であることを特定しました [17]。当時発行された唯一の CVE は Windsurf 用でしたが、攻撃パターンはエコシステム全体に適用されます。NIST はプロンプトインジェクションを「生成 AI の最大のセキュリティ欠陥」と特徴付け、OWASP はそれを LLM アプリケーション Top 10 での第 1 位の脆弱性としてランク付けしています [35]。

Claude Code には使用制限とコンピューティング制約があります。 2026 年初頭、多くの Pro プランユーザーは Sonnet を使用して 10〜15 分以内に使用制限に達し、Opus は開始した直後に利用不可 became と報告しました [27]。Anthropic はコンピューティングボトルネックへの対応として 2026 年 5 月に使用制限を倍増しましたが、根本的な制約は残っています：Claude Code は共有リソースであり、過剰な使用は最終的にレート制限に達します [39]。Pro プランには月額 $20 のエージェントクレジットが含まれ、Max 5x には $100、Max 20x には $200 が含まれます [2]。継続的な開発作業のために、ほとんどのチームは中断を防ぐために Max 5x 以上にアップグレードすることが必要であると発見しています。

Claude Code は人間のコードレビューを代替できません。 AI 生成の変更のすべては、

開発者が書いたコードと同じレビュープロセスを経るべきです。セキュリティ重要ファイルに触れる AI 生成の変更のすべては 2 人のレビュアーを必要とすべきです。エージェントはコラボレーターであり、自律的なシップパーではありません [20]。この原則は Stripe の経験によって強化されました：Stripe の開発者インフラストラクチャチームを率いる Scott MacVicar は、AI を代替ではなくコラボレーターとして扱うことが採用における最大の課題であると強調しました [8]。

ベンダーロックインとエコシステム依存性。 Claude Code は Anthropic モデルと最もよく動作するように設計され、Claude プラットフォームと密に統合されています。MCP のようなオープンスタンダードを使用していますが、Claude Code から別のエージェントや従来の開発への移行には現実的なコストがかかります：学習されたワークフロー、CLAUDE.md ファイル、カスタムスキル、フック、MCP サーバー設定はすべて Claude Code 固有です。チームは生産性向上がロックインリスクを上回るかどうかを評価すべきで、特に長いライフサイクルを持つプロジェクトにとって重要です。

これらの制限は Claude Code 自体の弱点ではありません。それは LLM ベースのエージェントに作業を委任するあらゆるシステムの特徴です。それらを理解することは、Claude Code を効果的かつ安全に使用する最初のステップです。

第 2 章：Claude Code の解剖

エージェント型ループ：収集、実行、検証

エージェント型ループの追跡：完全なターミナルウォークスルー

エージェント型ループを自動補完と異なるものにするもの

ループの中断と方向転換

背景にあるモデル：Sonnet、Opus、Fable、およびモデル選択

モデル選択：実践的な決定フレームワーク

ツール定義からのトークンオーバーヘッド

トークン管理数学：支出の理解

組み込みツール：ファイル操作、検索、実行、Web、コードインテリジェンス

.claude ディレクトリ：設定、CLAUDE.md、メモリ

コンテキストウィンドウ：どのように満たされるか、自動圧縮、トークン管理

コンテキストがいかに満たされるか：トークン予算のウォークスルー

自動圧縮がいかに動作するか

圧縮を生き残るものの制御

コンテキストを効果的に管理するための戦略

第 3 章：はじめるとセットアップ

プラットフォーム間のインストール：macOS、Linux、Windows、WSL、Homebrew、WinGet

Claude Code デスクトップアプリケーション

一般的なインストール問題のトラブルシューティング

認証とアカウントセットアップ

最初のセッション：ゼロからコードまでのウォークスルー

ターミナルセッショントレース：最初の対話

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

ステップバイステップ設定：.claudeignore、settings.json、権限モード

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

権限モード：デフォルト、自動受け取り、Plan、Auto モード

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

IDE 統合：VS Code、JetBrains、リモートコントロール、Web インターフェース

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

第 4 章：Claude Code のためのプロンプトエンジニアリング

Claude プロンプトの芸術：具体性、コンテキスト、検証

CLAUDE.md を永続的な指示レイヤーとして

Few-shot 例、XML 構造化、役割割り当て

なぜこれらの技術が機能するか：プロンプトエンジニアリングの背後にあるメカニクス

「私にインタビューして」パターン：Claude がコーディングの前に質問させる

プロンプトチェーンとセッション間ステート管理

一般的な失敗パターンと回避方法

敵対的テスト：**Claude Code** の出力のストレステスト

第 5 章：実践的な開発ワークフロー

フルスタックアプリケーションをゼロから構築（ハンズオンプロジェクト）

フェーズ 1：探索と仕様

フェーズ 2：実装

フェーズ 3：フロントエンド統合

フェーズ 4：実際のバグのデバッグ

フェーズ 5：敵対的テスト

実装中にコンテキスト圧縮を処理する方法

このワークフローがあなたに教えること

レガシーコードベースのリファクタリング：Stripe の Scala から Java への移行ケーススタディ

Stripe ケーススタディ：ディープダイブ

スケールでの移行ワークフロー

レビューボトルネック

複雑なバグのデバッグ：ステップバイステップウォークスルー

コードの大規模移行：**Wiz** の **Python** から **Go** への移行

馴染みのないコードベースへのオンボーディング

第 6 章：テスト、検証、品質保証

Claude に検証対象を与えること

テスト生成と自己修復テスト

Outcomes：自動化ルーブリック評価

Outcomes ルーブリックの記述：ステップバイステップウォークスルー

並列セッションを持つライター/レビュアーパターン

コードレビュースキルと敵対的レビュー戦略

CI/CD 統合：GitHub Actions、GitLab CI、自動化 PR

本番グレード CI/CD 構成

条件付き実行による自動化 PR レビュー

自動化プルリクエスト作成

CI/CD のセキュリティ考慮事項

コンプライアンスと監査要件

第 7 章：自動化とスケーリング

ヘッドレスモード：スクリプトとパイプラインのための非対話 Claude Code

ファンアウトパターン：何千ものファイルを並列処理

逐次ファンアウト（単純）

並列ファンアウト（高スループット）

結果集約を持つファンアウト

同時実行制御とバックオフを持つファンアウト

ファンアウトパターンのコスト分析

Claude Code ルーチン：スケジュール済みタスクとイベント駆動ワークフロー

Pre-commit フック、リンティング、自動化コード品質ゲート

コスト管理とトークン予算化のスケール

第 8 章：高度なエージェント型パターン

サブエージェント：調査と検証の委任

サブエージェントトークン経済学

スキル：再利用可能なドメイン知識とワークフローの作成

MCP サーバー：外部システム（GitHub、データベース、Jira）への接続

フック：セッション全体の自動化ポリシー強制

マルチエージェントオーケストレーション：リーダージェント、専門サブエージェント、共有ファイルシステム

マルチエージェントオーケストレーションの構成：ステップバイステップ

マルチエージェントワークフローでのマージ競合の解決

マルチエージェント実行からの可観測性データの解析

Claude マネージドア **rent** と「**Dreaming**」機能

Dreaming：自己改善エージェント

Outcomes：自動化品質評価

Claude Managed Agents の価格とデプロイメント

マルチエージェントオーケストレーション：パターンと失敗モード

Claude Finance と Add-ins

第 9 章：デプロイメントと本番運用準備

クラウドプラットフォームへの Claude Code 構築アプリケーションのデプロイ

環境設定とシークレット管理

モニタリングと可観測性：OpenTelemetry、使用量分析、トークン追跡

パフォーマンス最適化：プロンプトキャッシング、モデル選択、コンテキスト効率

プロトタイプから本番へ：Rakuten のケーススタディ

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

Rakuten の「AI-nization」戦略

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

技術的実装

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

スケールでの並列開発

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

将来の軌道：アンビエントエージェント

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

第 10 章：チームコラボレーションとエンタープライズ導入

スケールでの Claude Code：Stripe の 1,370 エンジニアデプロイメント

マネージド設定、エンタープライズセキュリティ、コンプライアンス

アイデンティティと API キーガバナンス

中央集権的トラフィックルーティング

MCP ツールガバナンス

ローカルサンドボックスと権限強制

エンドツーエンドエンタープライズ導入ウォークスルー

コンプライアンスフレームワーク

役割ベースアクセス制御

共有 **CLAUDE.md**、スキルライブラリ、プラグイン配布

既存チームワークフローへの **Claude Code** 統合

第 11 章：セキュリティディープダイブ

文書化された脆弱性：CVE-2025-59536 と CVE-2026-21852

プロンプトインジェクションと悪意のあるファイルコンテンツ

ファイル経由のプロンプトインジェクション

CI/CD プロンプトインジェクション

プライバシー安全装置

追加の安全装置

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

NIST と OWASP 分類

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

コンテキスト読み込みでのシークレット暴露

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

—dangerously-skip-permissions フラグ：いつどのように安全に使用するか

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

サンドボックス：Docker、Dev Containers、VM 隔離

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

エンタープライズセキュリティアーキテクチャ：MDM、LLM ゲートウェイ、エージェント監視

このコンテンツはサンプル本では読めません。この本は Leanpub で購入できます https://leanpub.com/claudecodejp.

第 12 章：エージェント型コーディングの未来

Anthropic のロードマップ：Dreaming、Outcomes、マルチエージェントオーケストレーション

エクスパートに関する研究：なぜドメイン知識がコーディング背景を打ち負かすか

2026 年以降のエージェント型コーディングを形成する 8 つのトレンド

ベンダーロックインとエコシステム依存性

ソフトウェアエンジニアの変化する役割

法的・倫理的状況

これが仕事の未来と知識経済にとって何を意味するか

第 13 章：結論：エージェント型エンジニアになること

初心者からマスターへの旅の統合

ソフトウェア開発の新しいメンタルモデル

Claude Code 実践の構築：個人アクションプラン

先を見据えて：次の 5 年が私たちをどこに連れて行くか

参考文献