WHITE PAPER

Claude Codeによる自律型開発エージェント入門

— AIがコードを理解し、計画・実装・テストまで進める新しい開発スタイル —



発行：Start Challenge Consulting co., ltd.
2026年8月

Start Challenge
Consulting co., ltd.
挑む力が、世界を照らす

Confidential ― 本資料の無断転載・二次配布を禁じます

はじめに：生成AI活用は「コードを書く」から「作業を進める」段階へ

開発支援AIの利用は、部分的なコード生成から、調査・計画・実装・検証までを一連で進める段階へ移行しつつある。Claude Codeは、この変化を開発現場で実際に運用するための選択肢として位置づけられる。

第1段階：コード補完

エディタ上での入力補完や候補提示が中心。対象は編集中のコードに限られ、判断はすべて開発者が行う。

→

第2段階：対話型のコード生成

チャット画面で質問し、生成されたコードを開発者が手元のファイルへ反映する。実行と検証は人が担う。

→

第3段階：開発作業の遂行

目的を伝えると、調査・計画・実装・テストまでを連続して進める。人は指示と承認に集中する。

本資料の対象読者と読みどころ

エンジニア

日々の実装・修正での使い方と、指示の具体化の方法

→ P7〜P12

開発リーダー

チームでの進め方、確認ポイントとレビュー体制の設計

→ P8・P13

PM・PMO

品質管理上の論点と、想定されるリスクの見立て

→ P14・P15

企業の推進担当者

導入判断に必要なガバナンスと効果測定の考え方

→ P15・P16

本資料は、Claude Codeを題材に「自律型開発エージェント」の考え方・使い方・運用上の留意点を整理したものである。特定製品の機能一覧ではなく、開発プロセスへの組み込み方を主題とする。

注記／事実：開発支援AIの利用形態は補完・生成・作業遂行の各形態が併存する。解釈：本資料では第3段階を「自律型開発エージェント」と定義する。

Claude Codeとは：開発環境の中で動くエージェント型コーディングツール

Claude Codeは、ターミナル上でプロジェクト全体のコードと設定を参照し、自然言語の指示から開発作業そのものを実行する。対話画面の中で完結するチャット型の生成AIとは役割が異なる。

提供元・位置づけ

Anthropicが提供する、エージェント型のコーディングツール。開発作業の遂行を目的とする。

動作環境

ターミナルを中心とした開発支援。既存の開発環境に組み込む形で利用する。

コードの理解範囲

プロジェクト内のソースコードや設定ファイルを横断的に調査し、文脈を踏まえて判断する。

指示の方法

自然言語で目的と条件を伝えることで、必要な開発作業を組み立てて実行する。

一般的なチャット型生成AIとの違い

一般的なチャット型生成AI

- やり取りは対話画面の中で完結する
- 参照できるのは、その場で提示した範囲のコードのみ
- 生成結果は開発者が手元のファイルへ反映する
- コマンド実行やテストは開発者が行う

Claude Code

- 開発環境（ターミナル）の中で動作する
- プロジェクト内のコードと設定を横断的に参照する
- 必要なファイルを直接編集する
- コマンドやテストの実行、結果に基づく修正まで含む

注記／事実：Claude CodeはAnthropicが提供するエージェント型コーディングツール。解釈：差異の本質は「情報提供」か「作業実行」かにある。機能仕様は提供元の最新ドキュメントで確認のこと。

自律型開発エージェント：目標から作業を分解し、検証まで連続して進める

自律型開発エージェントは、指示されたコードを部分的に生成するのではなく、目標を作業単位に分解し、調査・実装・テスト・修正までを連続して進める。結果として人間の役割は実装から監督・承認へ移る。

① 目標の受領

完了条件を含む目的を自然言語で受け取る

② 作業の分解

目的の達成に必要なタスクへ分割する

③ 関連ファイル調査

対象コードと依存関係を横断的に確認する

④ 実装方法の検討

変更方針と作業手順を整理し、提示する

⑤ コードの変更

複数ファイルにまたがる編集を実行する

⑥ テスト・修正

実行結果を確認し、必要に応じて修正する

エラーが解消するまで、調査・修正を繰り返す（③〜⑥の反復）

人間の役割は「作業者」から「監督者」へ移行する

目的と制約を定義する
何を、どこまで、どの条件で行うかを言語化して渡す。

計画と差分を確認する
実装前の計画と、実装後の変更内容を確認する。

品質基準を判断する
テスト結果と設計上の妥当性を最終判断する。

注記／解釈：自律性の度合いは指示と権限設定に依存する。提案：どの工程を人間の確認対象とするかを、導入時にあらかじめ定義すること。

従来のAIコーディングとの違い：対象範囲と担う役割が拡張される

違いは「どこまでをAIが担うか」にある。対象範囲は単一ファイルからリポジトリ全体へ、役割はコード補完からタスク遂行へ広がり、人間の作業は指示・確認・承認に集約される。

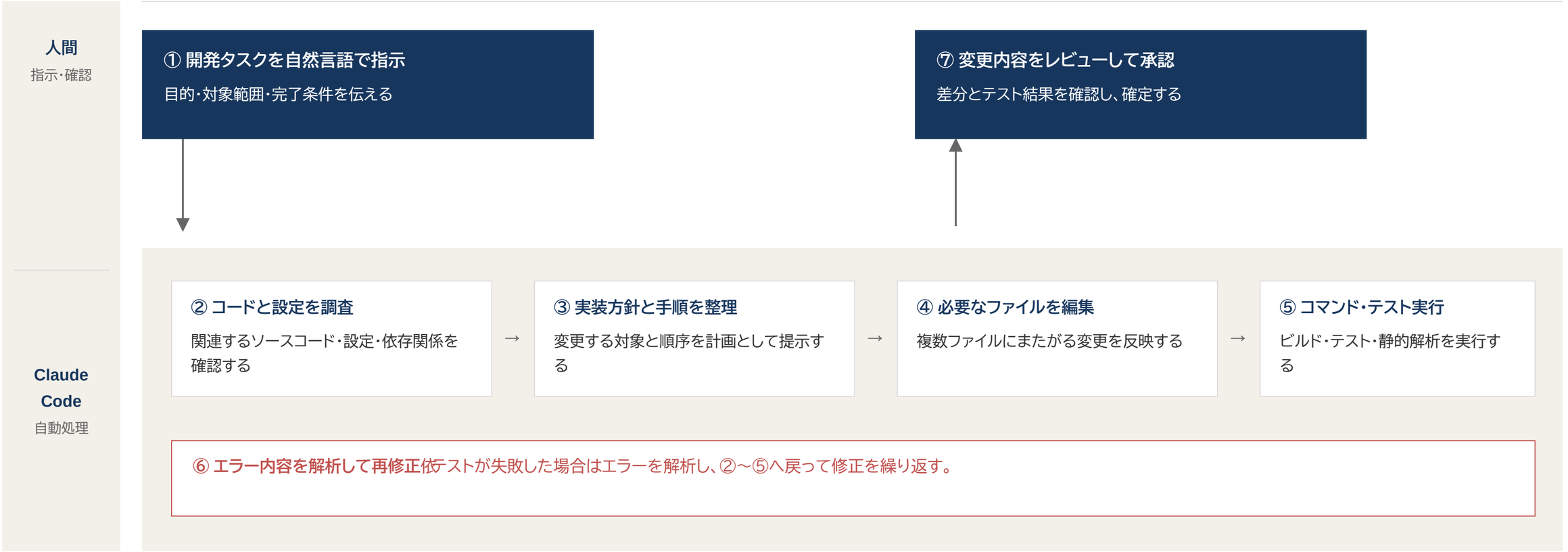
比較項目	従来のAIコーディング	自律型開発エージェント
主な役割	コード補完・質問回答	開発タスクの遂行
対象範囲	単一コード・単一ファイル	リポジトリ全体
作業計画	開発者が作成する	AIが分解し、計画として提案する
ファイル編集	生成結果の手動反映が中心	複数ファイルを横断して編集する
テスト	開発者が実行する	AIによる実行と、結果に基づく修正が可能
人間の役割	実装担当	指示・確認・承認

補足：役割が拡張されるほど、人間側に求められるのは「実装力」から「要件・制約・品質基準を定義する力」へ移る。

注記／解釈：両者は排他ではなく、補完的に併用される。表は役割の重心の違いを整理したもの。

基本的な仕組み：調査から再修正までを内部で循環させ、最後に人間が承認する

Claude Codeは、指示の受領から調査・方針整理・編集・実行・再修正までを一連の処理として循環させる。人間の関与点は、方針の確認と最終的な変更承認の二点に集約される。



人間が介在するのは①指示と⑦承認。②〜⑥は自動で循環するため、「どこで止めて確認するか」を事前に決めておくことが運用上の要点となる（詳細はP8）。

注記／解釈：図は処理の流れを概念的に整理したもの。実際の挙動・権限範囲は設定と利用形態により異なる。

Claude Codeでできること：実装だけでなく、調査・品質・保守の作業まで及ぶ

適用範囲は新機能の実装にとどまらず、原因調査・品質確保・保守作業にも及ぶ。特に、時間を要するが判断の難易度が高くない作業ほど、支援の効果が現れやすい。

A. 実装・変更

新機能の実装
要件から影響範囲を調べ、複数ファイルにまたがる変更を実装する。

既存コードのリファクタリング
重複処理や複雑な構造を整理し、動作を維持したまま構造を改善する。

ライブラリ更新に伴う修正
バージョン変更で影響を受ける箇所を洗い出し、追随して修正する。

開発環境・設定ファイルの調整
ビルド設定や実行環境の構成ファイルを、目的に沿って修正する。

B. 品質・検証

バグの原因調査と修正
再現条件とログをもとに原因箇所を特定し、修正案を作成する。

テストコードの作成
単体・結合テストや、不具合再発を防ぐ回帰テストを追加する。

コードレビュー支援
変更内容の観点整理や、規約との不一致の指摘を補助する。

Git操作・変更差分の整理
変更のまとまりを整理し、差分を確認しやすい形に整える。

C. 理解・ドキュメント

ドキュメント・コメントの整備
実装内容に基づき、READMEや処理コメントを整理する。

既存コードの構造把握
処理フローや依存関係を調べ、説明として出力する。

技術的負債の洗い出し
重複や複雑度の高い箇所を特定し、改善候補として整理する。

オンボーディング支援
新規参画者向けに、対象範囲の理解に必要な情報を整理する。

着手の順序：影響範囲が限定され、テストで検証しやすい作業（B・C領域）から始め、実装比重の大きい作業（A領域）へ段階的に広げる。

注記／解釈：分類は本資料による整理。提案：自社の開発工程に照らし、どの作業から適用するかを事前に選定すること。

基本的な開発フロー：8工程のうち3点に人間の確認ポイントを置く

依頼から確定までは8工程で構成される。全工程をAIに委ねるのではなく、④計画確認・⑧差分レビューを固定の確認ポイントとし、小さな単位で依頼することが品質確保の前提となる。

担当	工程	確認ポイント
人間	① **開発目的と完了条件を伝える**依何を、どこまで達成すれば完了かを明示する	
Claude Code	② **コードベースを調査する**依対象ファイル・依存関係・既存実装を確認する	
Claude Code	③ **実装計画を提示する**依変更対象と作業手順を、着手前に示す	
人間	④ **計画を確認する**依方針の誤りをこの段階で是正する	**確認ポイント 1** 仕様理解のずれをここで止める
Claude Code	⑤ **コードを変更する**依計画に沿って複数ファイルを編集する	
Claude Code	⑥ **テスト・静的解析を実行する**依変更が既存機能を壊していないか検証する	**確認ポイント 2** テスト結果の内容自体を人が確認する
Claude Code	⑦ **エラーを修正する**依失敗内容を解析し、②〜⑥を繰り返す	
人間	⑧ **差分をレビューして確定する**依変更内容と品質を判断し、承認する	**確認ポイント 3** 本番反映前の最終判断は人間が行う

依頼単位は小さく分割する。1回の依頼で扱う範囲が広いほど、④の計画確認と⑧の差分レビューの負荷が上がり、確認の実効性が低下する。

注記／提案：確認ポイントの位置と粒度は、対象システムの重要度に応じて設計する。

活用例① バグ修正：現象の入力から、原因特定・修正・回帰テストまで

バグ修正では、原因調査に要する時間が作業の大半を占めやすい。現象と再現条件を具体的に伝えることで、関連コード・ログ・テストの横断調査から回帰テストの追加までを一連で進められる。

1. 現象の入力

不具合の現象と再現条件、発生環境、期待する動作を伝える。

2. 調査

関連コード・ログ出力・既存テストを横断的に確認する。

3. 原因特定

原因となる処理を特定し、影響範囲とあわせて説明する。

4. 修正案の作成

修正方針を提示し、承認後にコードへ反映する。

5. 回帰テスト追加

同じ不具合の再発を検知するテストを追加する。

6. 結果の報告

テスト結果と変更点を整理して報告する。

指示例

「ログイン後に特定条件でセッションが切れる原因を調査し、修正してください。修正前に原因と影響範囲を説明してください」

要点：①調査対象を限定している依②修正前の説明を求めている依③着手前に方針を確認できる構造になっている

この工程で人間が確認すべきこと

- 提示された原因が、現象を過不足なく説明できているか
- 対症的な回避ではなく、原因そのものに対処しているか
- 修正の影響が、想定した範囲を超えて広がっていないか
- 追加された回帰テストが、再発を実際に検知できるか

留意点：原因の説明が曖昧なまま修正だけが提示された場合は、承認せず、根拠の提示を求める。

注記／解釈：工程は代表的な流れの整理。指示例は本資料による記述例。

活用例②　新機能開発：影響範囲の調査から、層をまたぐ実装とテストまで

新機能開発では、フロントエンド・API・DBにまたがる変更点の洗い出しが起点となる。要件から影響範囲を調査し、タスクを分割したうえで、完了条件に沿って検証まで進める。

進め方（6ステップ）

1. 要件から影響範囲を調査
既存実装のどこに、どのような変更が必要かを洗い出す。

2. フロントエンド・API・DBの変更点を整理
層ごとに必要な変更を分類し、依存関係を明らかにする。

3. 実装タスクを分割
レビュー可能な単位に分け、着手順序を決める。

4. 複数ファイルを横断して実装
分割した単位ごとに変更を反映する。

5. 単体テスト・結合テストを作成
正常系と異常系の双方について検証手段を用意する。

6. 完了条件に沿って検証
事前に定めた完了条件を満たしているかを確認する。

活用テーマ例

検索機能の追加
検索条件・並び順・ページングの設計を含む。

CSV出力機能の実装
文字コード・大量データ時の処理方式を検討する。

ユーザー権限管理の追加
権限区分と画面・API単位の制御を整理する。

外部APIとの連携
認証方式・失敗時の再試行方針を定める。

管理画面の改善
既存画面の構成変更と、影響する処理を確認する。

前提条件：新機能開発では「完了条件」を事前に文章化しておくことが不可欠となる。完了条件が曖昧な場合、実装は進むが期待した動作にならない、という結果を招きやすい。

注記／解釈：ステップとテーマ例は本資料による整理。実際の分割単位は対象システムの構成に依存する。

活用例③　既存システム：構造の把握から、負債の可視化と段階的改善へ

既存システムでは、仕様が文書化されておらず、構造の把握自体に時間を要する。理解・可視化・改善・文書化の順に進めることで、属人化した知識を組織で共有できる形に置き換えられる。

STEP 1　理解する

レガシーコードの構造把握

ディレクトリ構成や主要モジュールの役割を調査し、全体像を整理する。

処理フロー・依存関係の説明

特定機能について、呼び出し経路とデータの流れを説明として出力する。

→

STEP 2　可視化する

技術的負債の洗い出し

保守性を下げている箇所を列挙し、改善候補として整理する。

重複・複雑処理の特定

同種処理の重複や、分岐が過度に複雑な箇所を特定する。

→

STEP 3　改善する

段階的なリファクタリング

影響範囲の小さい単位から着手し、動作を保ったまま構造を改善する。

テストによる裏付け

改善の前後で挙動が変わらないことを、テストで確認する。

→

STEP 4　残す

仕様書・設計資料の作成

実装から読み取れる内容をもとに、README・仕様書を整備する。

引き継ぎ・オンボーディング

新規参画者が理解に要する時間を短縮する材料として活用する。

留意点：出力された「説明」は仕様の確定ではない

・出力されるのは「実装がそうなっている」という事実であり、「その動作が本来の仕様として正しい」ことを意味しない。
・既存の不具合や暫定対応も、そのまま仕様として記述される可能性がある。文書化した内容は、業務側の担当者による確認を経て確定させる。

注記／解釈：4段階の整理は本資料による。提案：文書化した仕様は、業務要件の所管部門によるレビューを必須とすること。

精度を高める指示の作り方：8つの要素を定型化し、依頼のばらつきを抑える

出力品質の差は、モデルの性能よりも指示の具体性に左右される。背景・範囲・制約・完了条件を定型のテンプレートとして共有することで、担当者による依頼品質のばらつきを抑えられる。

指示に含めるべき8つの要素

① 背景と目的を明確にする
なぜその作業が必要かを添える。判断の前提になる。

② 対象範囲を指定する
対象のディレクトリ・機能・画面を限定して伝える。

③ 使用技術・設計ルールを伝える
準拠すべき構成方針やコーディング規約を示す。

④ 変更してはいけない範囲を示す
影響を与えてはならないファイル・処理を明示する。

⑤ 完了条件を具体化する
どの状態になれば完了かを、検証可能な形で書く。

⑥ テスト方法を指定する
実行すべきテストコマンドと確認範囲を伝える。

⑦ 実装前に計画を提示させる
着手前に方針を確認する機会を必ず設ける。

⑧ 不明点があれば質問させる
推測で進めず、確認を求めるよう指示に含める。

指示テンプレート（雛形）

1. **目的** 何を実現したいか
2. **現在の問題** 発生している事象・課題
3. **対象範囲** 対象ファイル・機能
4. **制約条件** 変更禁止範囲・技術制約
5. **完了条件** 達成とみなす状態
6. **確認方法** 実行するテスト・検証手順
7. **報告事項** 作業前後に報告させる内容

赤枠の④⑤⑦は、手戻りの発生を抑える効果が特に大きい要素である。「変更してはいけない範囲」「完了条件」「事前の計画提示」の3点は、依頼のたびに省略しないことを運用ルールとする。

注記／解釈：8要素とテンプレートは本資料による整理。提案：テンプレートは組織で共有し、依頼フォーマットとして定着させること。

CLAUDE.mdによるルール設定：毎回伝える前提を、プロジェクトに常設する

プロジェクト固有の前提を毎回の指示に書き続けるのは現実的ではない。CLAUDE.mdに規約・テスト手順・禁止事項を記述して共有することで、担当者が変わっても同じ前提で作業を進められる。

記述する主な項目

技術スタック・ディレクトリ構成
使用言語・フレームワークと配置の方針

コーディング規約
命名・書式・設計上の約束事

実行すべきテストコマンド
変更後に必ず通すテスト・静的解析の実行方法

禁止する変更・操作
触れてはならないファイル、実行を認めない操作

ブランチ・コミットのルール
作業ブランチの切り方とコミット単位の方針

セキュリティ上の注意事項
認証情報の扱い、外部送信を禁じる情報の範囲

完了時の報告形式
変更内容・テスト結果・残課題を、どの粒度で報告させるか

運用上の注意

内容を増やしすぎない
網羅を目指して記述が膨らむと、重要なルールが埋もれる。守らせたい事項に絞る。

具体的な表現で書く
「丁寧に実装する」ではなく、判定可能な条件として記述する。

レビュー対象に含める
開発ルールと同様の管理対象とし、変更時はレビューを経る。

秘密情報を書かない
認証情報そのものは記載せず、取得方法のみを示す。

位置づけ：CLAUDE.mdは「プロジェクトの約束事」を明文化する場である

個々の依頼文（P12）が「今回の作業」を規定するのに対し、CLAUDE.mdは「このプロジェクトで常に守るべき前提」を規定する。両者を分けて管理することで、依頼文を簡潔に保てる。

注記／事実：CLAUDE.mdはプロジェクト固有の指示を記述・共有するためのファイル。解釈：記載項目の選定は本資料による整理。

利用時のリスクと注意点：品質・セキュリティ・運用の3領域で管理する

主要なリスクは品質・セキュリティ・運用の3領域に整理できる。いずれも「AIの出力をそのまま本番に反映しない」という原則を守ることで、影響を人間の確認工程の内側に留められる。

A. 品質に関するリスク

誤った仕様理解による実装

前提の解釈違いに気付かないまま実装が進むと、動作はするが要件を満たさない結果になる。

変更範囲が想定以上に広がる

関連箇所の整合を取ろうとして、依頼した範囲を超えた修正が行われる場合がある。

動作するが保守性の低いコード

テストは通るが、設計方針から外れた実装が蓄積し、将来の変更コストを押し上げる。

テスト不足による不具合の混入

検証の観点が不十分な場合、異常系や境界条件での不具合が見逃される。

B. セキュリティに関するリスク

機密情報・認証情報の取り扱い

APIキーや個人情報が、参照対象や出力に含まれないよう管理が必要となる。

外部ライブラリ・生成コードの安全性

追加される依存関係の妥当性と、既知の脆弱性の有無を確認する必要がある。

危険なコマンド実行の可能性

削除・上書き・外部通信を伴う操作は、実行可能な範囲を事前に制限する。

対象データ範囲の逸脱

本番データや権限外のリポジトリを参照させない運用が前提となる。

C. 運用で担保する対応

実装前に計画を確認する

仕様理解のずれを、コード変更前の段階で是正する。

依頼単位を小さく保つ

1回の変更範囲を限定し、差分を確認可能な量に抑える。

自動チェックを必須化する

テスト・静的解析・脆弱性検査を、承認の前提条件とする。

権限と操作範囲を制限する

参照可能なデータと実行可能な操作を、あらかじめ定義する。

人間によるレビューを経る

差分と品質の最終判断は人間が行う。

原則：AIの出力をそのまま本番環境に反映しない

生成された変更は「レビュー対象の提案」として扱う。テスト結果と人間の承認を経たうえで反映することを、例外なく適用する運用ルールとする。

注記／解釈：リスク分類および対応策は本資料による整理。提案：自社の開発標準・セキュリティ規程に照らして具体化すること。

企業導入に必要なガバナンス：範囲定義・承認工程・自動検証・効果測定

組織導入では、個々の使い方の工夫より仕組みの整備が効果を左右する。利用範囲の定義、承認工程、自動検証、効果測定の4つを整えることで、品質を保ちながら適用範囲を広げられる。

1. 利用範囲の定義

- ・利用可能なプロジェクトの指定
- ・扱えるデータの範囲の定義
- ・APIキー・秘密情報の管理方法
- ・AIに許可する操作範囲の設定
- ・参照を禁止する領域の明示

起点となる整備事項。ここが未定義のまま利用を広げると、統制が後追いになる。

2. 承認・履歴管理

- ・人間によるレビュー工程の設置
- ・承認者と承認基準の明確化
- ・Gitによる変更履歴の管理
- ・生成箇所の識別・追跡
- ・本番反映前の最終確認

既存の開発プロセスに組み込む形で運用し、例外を作らないことが要点。

3. 自動検証（CI/CD）

- ・CI/CDによる自動チェック
- ・静的解析の必須化
- ・脆弱性検査の実施
- ・テスト実行の必須化
- ・基準未達時のマージ禁止

人手のレビュー負荷を抑えつつ、品質の下限を機械的に担保する。

4. インシデント対応

- ・発生時の報告経路の整備
- ・切り戻し手順の事前定義
- ・影響範囲の特定方法
- ・原因分析と再発防止
- ・ルールへの反映プロセス

事象発生を前提に、検知から復旧までの手順を先に決めておく。

5. 効果測定指標（導入前にベースラインを取得しておく）

開発時間	修正回数	テスト通過率	レビュー指摘数	不具合発生率
同種タスクの着手から完了までの所要時間の変化	完了までに要した手戻り・再修正の回数	初回実行時にテストを通過した変更の割合	1件の変更あたりに指摘された件数の推移	リリース後に検出された不具合の件数・比率

測定の前提：導入前の実績値（ベースライン）を取得しておかなければ、効果を定量的に説明できない。指標は導入判断の前に確定させる。

注記／解釈：ガバナンス構成要素・指標の分類は本資料による整理。提案：既存の開発標準および情報セキュリティ規程との整合を確認のうえ適用すること。

まとめ：小さく始め、ルールを明文化し、任せる範囲を段階的に広げる

導入の成否を分けるのは、対象作業の選定とルールの明文化である。影響の限定された作業から着手し、確認工程を省略せずに運用しながら、任せる範囲を段階的に広げる進め方が現実的となる。

Claude Code導入のポイント

1. 小規模で影響の限定された作業から始める
失敗時の影響が小さく、検証しやすい領域を最初の対象とする。

2. 要件と完了条件を具体的に伝える
達成状態を検証可能な形で示し、解釈の余地を減らす。

3. プロジェクトルールを明文化する
規約・テスト手順・禁止事項をCLAUDE.mdに集約し、共有する。

4. 実装前の計画確認を習慣化する
着手前に方針を確認し、手戻りの発生源を抑える。

5. テストとコードレビューを省略しない

6. 任せる範囲を段階的に拡大する

導入チェックリスト

- [ ] 対象業務が決まっている
- [ ] コーディング規約が整理されている
- [ ] テストを自動実行できる
- [ ] 機密情報の管理ルールがある
- [ ] AIが操作できる範囲を定義している
- [ ] 人間による承認工程がある
- [ ] 導入効果を測定する指標がある

結論

Claude Codeの価値は、単なるコード生成の高速化ではない。調査・計画・実装・検証を一連の作業として支援し、開発者が設計や判断に集中できる環境をつくることにある。重要なのは、すべてをAIに任せるのではなく、人間が目的・制約・品質基準を管理することである。

注記／提案：チェックリストは導入判断前の確認事項として整理したもの。未充足項目は、着手前に整備対象として計画に含めること。

お問い合わせ：導入検討・適用範囲のご相談を承ります

対象業務の選定、指示テンプレートやCLAUDE.mdの整備、レビュー体制・ガバナンスの設計まで、導入段階に応じたご相談に対応いたします。

お問い合わせ先

会社名	**Start Challenge Consulting co., ltd.**
メールアドレス	**info@startchg.com**
担当	**平出**

お問い合わせの際は、対象システムの概要と検討段階（情報収集／試行／本格導入）をお知らせいただけますと、より具体的なご提案が可能です。

Start Challenge
Consulting co., ltd.
挑む力が、世界を照らす

ご相談いただける主なテーマ

- 対象業務の選定と試行計画の策定
- 指示テンプレートの整備と定着支援
- CLAUDE.mdおよび開発ルールの設計
- レビュー・承認工程の設計
- CI/CDによる自動チェックの整備
- 効果測定指標の設定と評価

Confidential ― 本資料の無断転載・二次配布を禁じます