

Claude Codeで既存システムを改修する方法

— コード解析から影響範囲の特定、実装・テスト・レビューまで —

Start Challenge Consulting co., ltd.

発行日：2026年8月

Confidential – 社外秘

既存システム改修に生成AIを活用する

既存システム改修の停滞は、コードを書く工程ではなく「現行仕様を読み解く工程」で発生している。Claude Codeを解析・影響範囲調査に充て、判断とレビューは人が担う進め方を前提とする。

既存システム改修が難しい理由

- 長期運用で仕様が積み上がり、全体像を一度に把握できない
- 変更の影響が他機能へ波及し、事前に範囲を見積もれない
- 改修に着手できる有識者が限られ、作業が集中する
- 止められない業務システムのため、失敗が許容されにくい

現場で起きていること

- 仕様書が最新化されておらず、正となる仕様がコードにしかない
- 当時の担当者が異動・退職し、設計意図が残っていない
- 確認作業が長期化し、改修そのものが後回しになる
- 結果として、影響の小さい範囲だけを継ぎ足す対応が続く

Claude影odeを活用する目的

- コード解析と影響範囲調査に要する工数を圧縮する
- 改修方針とテスト項目を、実装前に文書として可視化する
- 属人化した理解を、チームで共有できる形に置き換える
- 人はレビューと最終判断に時間を配分する

本資料で解説する内容

01
導入前の準備と権限設定

02
システム解析と影響範囲の特定

03
改修計画・実装・テスト

04
レビューとリスク管理

注記：解釈＝改修の停滞要因を「仕様理解の工程」と整理／提案＝解析工程へのAI活用と、人によるレビュー工程の維持

既存システム改修でよくある課題

> 改修が進まない主因は技術力の不足ではなく、現行仕様を読み解く情報が揃っていないことにある。特に上位3課題は、実装に着手する前の解析工程で解消できる。

課題 01

システム全体の構造が分からない

全体像を示す資料がなく、どのファイルから読み始めるべきか判断できない。着手前の調査だけで日数を要する。

起きること：見積り精度が上がらず、着手判断が先送りされる

課題 02

変更箇所の影響範囲を把握できない

共通処理や呼び出し元をすべて追い切れず、改修対象外の機能で不具合が顕在化する。

起きること：リリース後に想定外の障害が発生する

課題 03

コードと仕様書が一致していない

仕様書が更新されておらず、正となる仕様がコードにしか存在しない。資料を根拠にできない。

起きること：仕様確認のたびにコード読解が必要になる

課題 04

テストが不足している

自動テストが整備されておらず、改修前後の比較が手動確認に依存する。回帰範囲も定まらない。

起きること：品質の根拠を示せず、リリース判断に時間がかかる

課題 05

古い技術や独自実装が残っている

現行フレームワークの標準から外れた実装があり、一般的な知識だけでは読み解けない。

起きること：調査対象が広がり、修正方針を決めきれない

課題 06

改修できる担当者が限られている

有識者に作業とレビューが集中し、属人化が解消されない。並行して改修を進められない。

起きること：改修待ちが常態化し、要望が滞留する

注記：解釈＝赤帯（課題01〜03）は解析工程で解消しうる情報不足の課題、灰帯（課題04〜06）は体制・資産に起因する構造的課題として区分

Claude影odeとは

Claude Codeはターミナル上で動作し、リポジトリ内の複数ファイルを横断して調査・修正・テストまで扱える。貼り付けたコードしか見えないチャット型AIとの差は、既存システム改修で最も大きく表れる。

ツールの特徴

- ターミナル上で利用するAIコーディング支援ツール。開発者の手元の環境で動作する
- プロジェクト内の複数ファイルを横断して読み取り、構造や依存関係を分析できる
- コードの検索・修正・テスト実行までを、対話しながら段階的に進められる
- 既存の実装や命名規則を読み取ったうえで、それに沿った変更を提案できる

比較の観点	一般的なチャット型生成AI	Claude影ode
利用場所	ブラウザ上のチャット画面	開発者のターミナル（プロジェクト配下）
参照できる情報	貼り付けた範囲のコードのみ	リポジトリ内の複数ファイルを横断
作業の単位	単発の質問と回答	調査・修正・テスト実行までの一連の作業
前提の共有	都度、背景を説明する必要がある	既存の実装・規約を読み取って前提化できる
改修との相性	部分的な相談・記述支援に向く	**全体把握と影響範囲調査に適する**

既存システム改修との相性

資料がなくても着手できる

仕様書が古くても、コードそのものを一次情報として構造と処理内容を整理できる。

横断検索で漏れを減らす

呼び出し元・共通部品・設定ファイルまで、影響範囲の候補を機械的に洗い出せる。

工程を分けて指示できる

「調査のみ」「計画のみ」と作業を区切れるため、実装前に人が判断を挟める。

前提となる制約

出力は仮説を含む。妥当性の確認と最終判断は人が行う前提で組み込む。

注記：本ページの記載は本資料の想定する運用方法の整理。実際に利用可能な機能・操作権限は、利用環境および契約条件により異なる

Claude Codeで対応できる改修業務

対応業務は「理解する」「変更する」「品質を担保する」の3領域に整理できる。効果が最も安定するのは理解の領域であり、変更の領域は人の承認を挟むことを前提とする。

領域A：理解する

解析・調査

01 システム構成の把握

ディレクトリ構成・使用技術・処理の起点を整理させる

02 処理内容の説明

対象機能の処理フローを、条件分岐を含めて言語化させる

03 影響範囲の調査

呼び出し元・共通部品・設定値を横断検索で洗い出させる

04 不具合原因の特定

再現条件とログから、原因箇所の候補を絞り込ませる

領域B：変更する

実装

05 機能追加・仕様変更

既存の設計・命名規則に沿った形で差分を作成させる

06 リファクタリング

対象と範囲を明示したうえで、構造の整理を限定的に行わせる

この領域の前提

変更対象ファイルと方針を人が承認したうえで着手する。1回の指示で複数機能を変更させない。

領域C：品質を担保する

検証・記録

07 テストコードの作成

正常系・異常系・境界値を対象にテストを追加させる

08 ドキュメントの更新

改修内容をもとに仕様書・運用手順の差分案を作成させる

期待できる効果

属人化していた仕様理解が文書として残り、次回改修の着手時間を短縮できる。

注記：解釈＝業務を「理解／変更／品質担保」の3領域に区分／提案＝領域Bは人の承認を前提工程として組み込む

Claude Code導入前に準備すること

準備は「環境」「品質基準」「情報統制」の3区分で整える。特に機密情報の取り扱いと許可する操作範囲は、利用開始前に組織として決定しておく必要がある。

区分	準備項目	確認する内容	未実施の場合のリスク
環境	対象リポジトリの準備	改修対象のコードが手元に揃っているか。関連リポジトリが分割されていないか	解析範囲に漏れが生じ、影響範囲を誤る
	開発環境・実行環境の確認	ローカルでビルド・起動できるか。依存ライブラリを取得できるか	修正の妥当性を実行して確認できない
	Gitブランチの作成	改修用ブランチを分離しているか。差分の確認と切り戻しが可能か	意図しない変更を元に戻せない
品質基準	テスト実行方法の確認	既存テストの実行手順と、改修前時点での成否を把握しているか	改修が原因の失敗かを切り分けられない
	コーディング規約の整理	命名規則・構成ルール・禁止事項を提示できる形にしているか	既存コードと様式が異なる変更が混入する
情報統制	機密情報・個人情報の確認	認証情報・接続情報・個人情報が対象範囲に含まれていないか	秘匿すべき情報が外部へ渡る
	許可する操作範囲の設定	ファイル変更・コマンド実行・外部接続のうち、どこまで許可するかを定めているか	想定外の環境・資源に操作が及ぶ

着手条件　「切り戻せる状態（ブランチ）」「改修前のテスト結果」「許可する操作範囲」の3点が揃うまで、Claude Codeによるコード変更は開始しない。

注記：提案＝準備項目を「環境／品質基準／情報統制」に区分し、情報統制2項目は組織としての事前決定事項と位置づける

既存システム改修の基本フロー

改修は8ステップに分解し、工程ごとに指示を切り分ける。実装前（STEP5）と反映前（STEP8）に人の判断を必ず挟むことで、AIの推測がそのまま本番へ到達する経路を遮断する。

フェーズ 1：理解する（調査）

STEP 1
改修目的を整理する
何を、なぜ変えるのかを言語化し、完了条件まで先に定義する

STEP 2
システム全体を把握する
構成・使用技術・主要な処理フローをClaude Codeに整理させる

STEP 3
関連コードを特定する
対象機能を構成するファイルと処理の入口を絞り込む

フェーズ 2：決める（計画）

STEP 4 最重要
影響範囲を調査する
呼び出し元・共通部品・データベースへの波及を洗い出す

STEP 5
改修方針を作成する
変更対象ファイル・変更内容・想定リスク・テスト項目を一覧化する

フェーズ 3：変える・確かめる（実装〜反映）

STEP 6
小さな単位で修正する
1指示1目的とし、変更のたびに差分を確認する

STEP 7
テストを実施する
既存テストの再実行と、追加テスト・回帰テストを行う

STEP 8 最終判断
人がレビューして反映する
差分・要件充足・セキュリティを確認し、反映可否を人が決定する

工程を分ける理由

推測の混入を検出できる
調査結果を文書で受け取ることで、事実と推測を人が切り分けられる。

手戻りを小さくできる
方針段階で誤りを修正すれば、実装後の作り直しが発生しない。

レビュー範囲を限定できる
変更単位が小さいほど、差分レビューの精度と速度が保たれる。

説明責任を果たせる
各工程の記録が残り、改修判断の根拠を後から提示できる。

注記：提案＝8ステップを3フェーズに区分し、STEP5（方針承認）とSTEP8（反映判断）を人の判断ポイントとして固定

ステップ1：改修要件を具体化する

要件が曖昧なままでは、AIは不足部分を推測で補い、変更範囲が想定外に広がる。「変更してはいけない仕様」と「完了条件」を先に固定することが、改修範囲を制御する最も有効な手段となる。

具体化の5観点

1. **現在の動作と期待する動作を分ける**
   「今どう動くか」と「どう動いてほしいか」を別々の文で記述する
2. **対象ユーザー・対象画面を明確にする**
   権限や画面によって挙動が異なる場合、その条件まで指定する
3. **正常系・異常系を整理する**
   入力エラー・権限不足・通信失敗時の期待挙動も要件に含める
4. **変更してはいけない仕様を明示する**
   既存の画面・API・データ構造のうち、維持すべきものを列挙する
5. **完了条件を設定する**
   何が確認できれば改修完了とみなすかを、検証可能な形で定義する

要件整理例（承認機能の追加）

対象機能	申請一覧画面および申請登録API
現在の問題	申請が登録と同時に確定し、上長による確認工程が存在しない
変更内容	「承認待ち」状態を追加し、承認権限を持つ利用者のみ確定できるようにする
制約条件	既存の申請登録APIの入出力形式は変更しない／既存データの状態値は移行対象外とする
完了条件	承認前は確定されず、権限のない利用者が承認操作を行えないこと
確認方法	単体テスト（正常系・権限エラー）と、既存の申請系テストの全件再実行

曖昧な要件が引き起こすこと

仕様の推測
記載のない挙動を一般的な実装例から補完し、現行業務と異なる仕様が実装される。

変更範囲の拡大
改善と判断した箇所まで同時に修正され、レビュー対象の差分が肥大化する。

既存仕様の破壊
維持すべき入出力形式が変更され、連携する他機能で不具合が発生する。

対策
「制約条件」と「完了条件」を要件票として先に確定し、指示文に必ず含める。

注記：要件整理例は説明用に作成した架空の事例であり、特定システムの仕様ではない

ステップ2：システム全体を解析する

解析は対象機能から始めず、全体構造の把握から着手する。外側（構成）から内側（処理フロー）へ順に降りることで、改修対象がシステム内のどこに位置するかを地図として共有できる。

順	解析項目	Claude Codeへの依頼内容
1	ディレクトリ構成を確認する	主要ディレクトリの役割を階層ごとに整理させる
2	使用言語・フレームワークを把握する	依存定義ファイルから言語・FW・主要ライブラリと版を一覧化させる
3	アプリケーションの起点を特定する	起動処理・ルーティング定義・初期化処理の場所を示させる
4	画面・API・DBの関係を整理する	画面ごとに呼び出すAPIと参照テーブルを対応表として出力させる
5	外部サービスとの連携箇所を確認する	外部通信を行う処理と、その接続設定の定義場所を洗い出させる
6	主要な処理フローを説明させる	代表的な業務処理を、入口から結果保存まで順を追って説明させる

解析の進め方（外側から内側へ）

レイヤー1：全体構成
ディレクトリ構成／使用技術／実行環境

レイヤー2：入口と経路
起動処理／ルーティング／画面とAPIの対応

レイヤー3：データと外部接点
テーブル参照・更新箇所／外部サービス連携

レイヤー4：処理フロー
条件分岐・例外処理を含む業務処理の流れ

この段階で作成しておく成果物

構成メモ
ディレクトリと役割の対応表

機能マップ
画面・API・テーブルの対応表

連携一覧
外部接続先と設定の定義場所

解析結果の受け取り方

出力には、コードから読み取れた事実と、AIによる推測が混在する。「根拠となるファイル名と該当箇所を併記する」「不明な点は不明と明記する」ことを指示し、根拠のない記述は仕様として採用しない。

注記：提案＝解析を4レイヤーに分け、外側から内側へ段階的に降ろす進め方を推奨／成果物は次工程（影響範囲特定）の入力として使用する

ステップ3：影響範囲を特定する

改修事故の多くは、改修対象そのものではなく共通部品を経由した波及によって発生する。呼び出し元・共通処理・データ更新箇所の3方向を洗い出し、7領域のチェックで漏れを検出する。

順	調査項目	確認する観点
1	対象機能を呼び出しているコードを検索	対象の関数・クラス・エンドポイントの参照元をすべて列挙する
2	関連する画面・API・バッチを確認	同一処理を利用する画面や、夜間バッチからの利用がないか確認する
3	共通部品への影響を確認	共通処理を変更する場合、他機能への波及範囲を必ず個別に確認する
4	DBの参照・更新箇所を調査	対象テーブルを参照・更新している処理と、制約・既存データを確認する
5	設定ファイルや環境変数を確認	環境ごとに値が異なる設定に依存していないかを確認する
6	既存テストから関連仕様を読み取る	テストコードに記述された期待値を、現行仕様の根拠として整理する

影響範囲チェック（7領域）

全領域で有無を判定

フロントエンド	画面表示・入力制御・権限による表示差
バックエンド	業務ロジック・API入出力形式
データベース	項目追加・制約・既存データの整合
外部API	連携先への送信内容・受信処理
認証・権限	操作可否の判定・権限の付与範囲
バッチ処理	夜間処理・定期実行からの利用
ログ・監視	出力内容の変化・監視条件への影響

波及の3方向

上流へ（呼び出し元）
この処理を使っている画面・API・バッチ

横へ（共通部品）
同じ共通処理を利用する他の機能

下流へ（データ）
更新されるテーブル・外部連携・出力ログ

この工程の完了条件

7領域すべてについて「影響あり／影響なし／要確認」を判定し、判定の根拠となるファイル名を併記した一覧が作成されている状態。

注記：提案＝影響範囲を7領域に分解し、AIの検索結果に対して領域単位で人が判定を確定する運用とする

ステップ4：改修計画を作成する

実装の前に「変更対象ファイル一覧」「想定リスク」「テスト項目」を文書化し、人が承認する工程を挟む。この一手間が、修正後に発覚する手戻りを最小化する。

計画作成の6原則

いきなりコードを修正させない
この段階では「変更しない」ことを明示的に指示する

変更対象ファイルを一覧化する
対象と対象外の境界を先に確定させる

ファイルごとの変更内容を整理する
何をどう変えるかを行単位まで具体化する

想定リスクを洗い出す
他機能への波及とデータ影響を明記する

テスト項目を先に決める
合格基準を実装前に固定する

改修計画を人がレビューする
承認されるまで実装工程へ進めない

改修計画書の構成（変更対象ファイル一覧）

対象ファイル	変更内容	想定リスク	テスト
申請登録処理	状態値に「承認待ち」を追加	既存データの状態判定	単体・回帰
申請一覧画面	承認操作ボタンの表示制御を追加	権限判定の漏れ	単体・画面
権限判定共通処理	承認権限の判定条件を追加	他機能へ波及	全機能回帰
夜間集計バッチ	集計対象から承認待ちを除外	集計値の変動	結果比較

計画レビューで人が確認する4点

確認 01
対象範囲の妥当性
要件に対して変更対象が多すぎないか。要件外のファイルが含まれていないか。

確認 02
共通処理の変更有無
共通部品を変更する計画であれば、代替案（呼び出し側での対応）を検討する。

確認 03
前提の根拠
計画中の仕様理解が、コード上の根拠に基づいているか。推測が含まれていないか。

確認 04
テストの十分性
影響範囲7領域のうち、影響ありと判定した領域が全てテスト項目に対応しているか。

承認ゲート　改修計画の承認をもって実装工程へ進む。計画外のファイル変更が必要と判明した時点で、実装を中断し計画へ差し戻す。

注記：計画書の記載例は説明用に作成した架空の事例であり、特定システムの仕様ではない

ステップ5：小さな単位でコードを修正する

1回の指示で変更する対象を1目的に限定し、差分確認と切り戻し可能性を常に維持する。変更単位が小さいほど、原因の切り分けとレビューの精度が保たれる。

実装時の6原則

範囲の限定	**1回の指示で複数機能を変更しない** 変更が失敗した場合の原因を1つに絞れる状態を保つ
様式の維持	**既存の設計・命名規則を維持する** 周囲のコードと同じ書き方に揃え、可読性の断絶を作らない
差分の抑制	**不要なリファクタリングを避ける** 整形・改名・構造変更を同一の変更に混在させない
意図の記録	**修正理由が分かるコメントを残す** なぜその実装にしたかを、次の担当者へ引き継ぐ
逐次の確認	**変更ごとに差分を確認する** 計画にないファイルが変更されていないかを毎回確認する
復旧の担保	**問題があればすぐ戻せる状態にする** 変更単位ごとに記録を残し、切り戻し先を明確にする

観点	大きな単位で変更した場合	小さな単位で変更した場合
原因の切り分け	どの変更が原因か特定できない	直前の変更に原因を限定できる
レビュー精度	差分が多く、見落としが生じる	全差分を確認できる分量に収まる
切り戻し	必要な変更まで巻き戻る	問題箇所のみを戻せる
テスト	失敗の要因が複合する	対象範囲を都度確認できる
進捗の可視性	完了か未完了かしか分からない	どこまで確認済みかを示せる

1変更サイクルの進め方

01 **対象を1つ指定**
計画書の1行分だけを指示する

02 **修正を実行**
既存の様式に沿って差分を作成させる

03 **差分を確認**
計画外の変更が混入していないか点検する

04 **動作を確認**
対象範囲のテストを実行し結果を残す

05 **記録して次へ**
変更単位で記録し、戻せる状態を作る

注記：解釈＝変更単位の大小による差異を観点別に整理／提案＝計画書1行を1サイクルとして実装を進める運用とする

Claude Codeへの指示例

同じ改修依頼でも、指示の粒度が結果を決める。「調査までで止める」「変更してはいけないものを示す」「承認後に実装する」の3点を明文化することで、成果物を人が検証できる形に保てる。

悪い指示例

「このシステムに承認機能を
追加してください」

- 対象範囲が示されず、変更が全体へ広がる
- 調査工程が省略され、推測で実装される
- 維持すべき仕様が不明で、既存機能が壊れる
- 人が検証する前に差分が完成してしまう

改善した指示例

1. まず関連するコードを調査する
2. **この段階ではファイルを変更しない**
3. 変更候補ファイルと理由を一覧化する
4. 既存機能への影響を整理する
5. 改修方針とテスト項目を提示する

指示に含める6要素

- **背景** ／ なぜこの改修が必要か
- **対象範囲** ／ 対象の画面・API・ファイル
- **期待する動作** ／ 正常系と異常系
- **制約** ／ 維持すべき仕様・規約
- **禁止事項** ／ 変更・削除してはならない対象
- **完了条件** ／ 何が確認できれば完了か

工程	指示に必ず入れる一文	受け取る成果物
調査	「この段階ではファイルを変更せず、根拠となるファイル名と該当箇所を併記して報告する」	関連ファイル一覧・処理フローの説明
計画	「変更対象ファイル・変更内容・想定リスク・テスト項目を一覧で提示し、承認まで実装しない」	改修計画書（変更対象一覧・リスク・テスト項目）
実装	「承認済みの計画のうち1件のみを対象とし、既存の命名規則に従い、他の整形は行わない」	1目的に限定された差分・変更理由のコメント
テスト	「正常系・異常系・境界値を対象にテストを追加し、実行結果と未確認項目を明記する」	テストコード・実行結果・未確認項目の一覧

注記：指示文は説明用に作成した例であり、実際の運用では対象システムの規約・制約に合わせて記述を調整する

ステップ6：テストを作成・実行する

改修前の時点でテストを実行し、その結果を基準として保持する。基準がなければ、テスト失敗が改修起因か既存の問題かを切り分けられず、品質の根拠を示せない。

改修前：基準を作る

01
既存テストを先に実行する
改修に着手する前に全件実行し、成功・失敗の状態を記録する。既に失敗しているテストは改修対象外として区別する。

02
改修前の動作を確認する
対象機能の現状の挙動を、入力と出力の組で記録する。改修後の比較対象として使用する。

改修後：差分を確かめる

03
単体テストを追加する
計画段階で決めたテスト項目に対応するテストを追加する。項目と実装の対応関係を明示する。

04
正常系・異常系・境界値を確認する
権限不足・入力不正・上下限値など、業務上起こりうる条件を網羅する。

05 影響範囲7領域と対応
回帰テストを実施する
STEP3で「影響あり」と判定した領域について、改修対象外の機能が従来どおり動作することを確認する。共通処理を変更した場合は、利用する全機能を対象とする。

06 レビューへの引き継ぎ
テスト結果と未確認項目を記録する
「確認済み」と「未確認」を明示的に分けて記録する。未確認項目を残したまま完了とせず、残存リスクとしてレビュー時に提示する。

テスト種別	確認する内容	実施タイミング	記録すること
既存テスト（基準取得）	改修前の成功・失敗状態	改修着手前（必須）	既存の失敗テスト一覧
単体テスト（追加）	追加・変更した処理の期待動作	変更単位ごと	計画のテスト項目との対応
回帰テスト	改修対象外の機能が従来どおり動作すること	全変更の完了後（レビュー前）	改修前結果との差異

注記：提案＝改修前のテスト実行結果を「基準」として記録し、改修後の結果との差異のみを評価対象とする運用とする

ステップ7：コードレビューと品質確認

> レビューはAIの説明文ではなく、Git差分そのものを対象とする。「要件との一致」「範囲の妥当性」「セキュリティ」の3点を人が確認し、反映可否を判断する。

順	確認項目	具体的に見るところ
1	Git差分を人が確認する	変更ファイル・行単位の差分をすべて目視する。生成された説明で代替しない
2	要件どおりの変更か確認する	STEP1の完了条件・制約条件に対して、過不足なく実装されているか
3	変更範囲が必要以上に広くないか確認する	計画外のファイル変更・整形のみの差分・意図しない削除が含まれていないか
4	セキュリティ上の問題を確認する	権限判定の抜け・入力値検証の欠落・認証情報のコード内記述がないか
5	エラーハンドリングを確認する	例外を無視していないか。失敗時に処理が中途半端な状態で残らないか
6	ログ出力や性能への影響を確認する	出力内容に機密が含まれないか。ループ内の問い合わせなど処理量の増加がないか
7	AIが生成した説明だけで判断しない	「テストは成功しました」等の報告は、実行結果と差分を自ら確認して裏付ける

レビューの役割分担

AIに任せられること

- 差分の要約と変更理由の説明
- 実装漏れの候補の指摘
- テストケースの追加案の提示

人が必ず担うこと

- 要件との一致の最終判断
- セキュリティ・権限の妥当性判断
- 本番反映の可否決定と説明責任

判断を委ねてはならないこと

テスト成功の報告のみを根拠とした品質判定。および、不要と判断されたコードの削除可否。

反映可否の判定基準

計画との一致
変更ファイルが承認済み計画の範囲内に収まっている

テストの充足
影響ありと判定した領域の回帰テストが完了している

残存リスクの明示
未確認項目が一覧化され、関係者に共有されている

切り戻し手段
障害発生時に戻す手順と判断者が決まっている

注記：提案＝レビュー対象をGit差分に固定し、AI生成の説明文は補助情報として扱う運用とする

既存システム改修で注意すべきリスク

リスクは「AIの推測に起因するもの」と「運用ルールの不備に起因するもの」に大別される。特に認証情報の取り扱いと本番環境の操作は、技術的対策ではなく規程として禁止すべき領域である。

区分	リスク事象	想定される影響	対策
AIの推測に起因	存在しない仕様を推測する	現行業務と異なる仕様が実装される	根拠となるファイル名と該当箇所の併記を必須とし、根拠のない記述を仕様として採用しない
	使用されていないように見えるコードを削除する	動的に呼ばれる処理やバッチ経由の利用が停止する	削除は改修対象と切り離し、別途調査と承認を経た上で単独の変更として実施する
	共通処理を変更して別機能に影響を与える	改修対象外の機能で不具合が発生する	共通処理の変更は原則避け、呼び出し側での対応を優先。やむを得ない場合は全利用機能を回帰対象とする
	ライブラリを安易に更新する	互換性が失われ、広範囲の修正が必要になる	依存定義の変更を禁止事項として明示し、更新は独立した案件として計画する
運用ルールの不備に起因	テスト成功だけで品質を判断する	テスト未整備の範囲の不具合が検出されず本番に出る	テスト網羅範囲と未確認項目を明記し、差分レビューと併せて反映可否を判断する
	認証情報をプロンプトやコードへ含める	秘匿情報の漏洩、および不正利用	対象範囲から機密情報を分離し、規程として記載を禁止。混入時の失効手順を定める
	本番環境を直接操作させる	業務停止・データ破損など回復困難な事象	許可する操作範囲を検証環境に限定し、本番接続情報を作業環境から排除する

推測系リスクの共通対策

「変更しない」「削除しない」「更新しない」対象を、指示文の禁止事項として毎回明示する。

運用系リスクの共通対策

機密情報の分離と本番環境の隔離は、個人の判断に委ねず組織の規程として定める。

注記：解釈＝リスクを「AIの推測に起因」「運用ルールの不備に起因」に区分／提案＝深紅表示の2項目は規程による禁止事項として扱う

Claude Code活用チェックリスト

改修前・改修中・改修後の3時点で確認項目を固定し、案件ごとの判断のばらつきを抑える。改修後の4項目が完了していない状態を「改修完了」と扱わない。

改修前　着手判断（4項目）

- ☐ **改修目的と完了条件が明確か**
  何が確認できれば完了かを文書化している
- ☐ **対象範囲を限定しているか**
  対象と対象外の境界、変更禁止対象を明示した
- ☐ **バックアップ・ブランチを用意したか**
  切り戻し先が確保されている
- ☐ **既存テストを実行したか**
  改修前の成否を基準として記録した

改修中　実装管理（3項目）

- ☐ **影響範囲を確認したか**
  7領域すべてで影響の有無を判定した
- ☐ **変更差分を都度確認しているか**
  1変更ごとに差分を目視している
- ☐ **不要な変更が含まれていないか**
  整形のみの差分・計画外ファイルがない

中断の判断
計画外のファイル変更が必要と判明した時点で実装を止め、改修計画へ差し戻す。

改修後　完了判定（4項目）

- ☐ **回帰テストを実施したか**
  改修前の結果と差異を比較した
- ☐ **人によるレビューを完了したか**
  Git差分を人が確認し反映可否を判断した
- ☐ **仕様書や運用手順を更新したか**
  次回改修のための記録を残した
- ☐ **障害発生時の切り戻し方法があるか**
  手順と判断者が決まっている

注記：提案＝赤枠の4項目（切り戻し確保・影響範囲判定・人のレビュー・切り戻し手順）を、案件ごとに省略不可の必須項目とする

まとめ

Claude Codeの効果は「解析と影響範囲調査の短縮」に集約される。工程を分けた指示・小さな変更単位・人によるレビューを開発プロセスとして定めることが、安全な改修の前提条件となる。

活用の要点

- Claude Codeは既存コードの理解と改修を効率化できる
- 最初にコード解析と影響範囲調査を行う
- 効果が最も安定するのは、実装ではなく理解の工程

進め方の要点

- 「調査・計画・実装・テスト」を分けて指示する
- 小さな変更単位で差分を確認する
- 計画の承認をもって実装に進む

品質責任の所在

- 最終判断と品質保証は人が担当する
- AIの説明のみを根拠に反映を決定しない
- 機密情報の分離と本番環境の隔離は規程で定める

組織としての要点

- AI任せにせず、レビュー工程を含めた開発プロセスを整備する
- 解析結果を文書として残し、属人化を解消する
- チェックリストを標準手順として運用する

安全な改修プロセスの全体像

要件の具体化 — 全体解析 — 影響範囲の特定 — 計画の承認 — 小単位の実装 — テスト・回帰 — 人のレビューと反映

意思決定事項

Claude Codeの適用は「解析・影響範囲調査」から開始し、実装工程は計画承認と人のレビューを必須のゲートとして運用する。

決定 01	決定 02	決定 03	決定 04
適用範囲を解析・調査工程から開始する	計画承認前の実装着手を禁止する	機密情報の分離と本番環境の隔離を規程化する	チェックリストを標準手順として定着させる

注記：提案＝本資料で整理した進め方を標準プロセスとして定義し、案件単位ではなく組織の開発規程として運用することを推奨する

お問い合わせ

既存システムの解析・改修計画の策定から、生成AIを前提とした開発プロセスの整備まで支援します。現状の課題整理からのご相談も承ります。

お問い合わせ先

会社名	Start Challenge Consulting co., ltd.
メール	info@startchg.com
担当	平出

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

既存システムの改修方針・生成AI活用の
ご相談を承っております

ご相談いただける内容

01
既存システムの構造解析と
ドキュメント整備

02
影響範囲調査と
改修計画の策定支援

03
レビュー工程を含む
開発プロセスの整備

04
生成AI活用時の
情報統制ルールの設計