Givery, Inc.

ORIENTATION · 2026 年 6 月版

# オリエンテーション 座学編

## 生成AI実務研修（7 時間 × 2 日間）
## 協栄産業株式会社様向け

| | |
|---|---|
| **提供** | Givery 株式会社 / 講師 安田 光喜 |
| **配布先** | 協栄産業株式会社 御中 |
| **所要** | Day 1 S01 座学 [60min] の投影資料 兼 2 日間の概念リファレンス |
| **環境** | VS Code × Claude Code CLI（統合ターミナル経由） |
| **版数** | v2.0（2026 年 6 月 20 日） |

Day 1 冒頭の座学 [60min]（AI 最新動向と Claude Code 基礎）で投影する資料であり、2 日間を通じて参照する概念リファレンスです。Skills / Subagents / Commands / Hook / MCP の使い分け、仕様駆動開発、テスト駆動開発、セキュリティの線引きまでを 1 冊に収めています。手を動かす手順は「ハンズオンガイド Day 1 / Day 2」を参照してください。

# 1. 本書の位置付けと 2 日間のゴール

本研修は「Claude Code を主軸とした AI 駆動開発」を 2 日間で手の内に入れる研修です。Day 1 はほぼ終日が手を動かす演習で、Claude Code の基礎操作から、Skills / Subagents / Commands という道具立て、仕様駆動開発の入り口までを順に踏みます。Day 2 は仕様駆動開発を完走し、Kiro 式との比較、テスト駆動開発を経て、題材「技術営業支援 AI エージェント」を作り切ります。本書は Day 1 冒頭の座学 [60min] で投影する資料であり、2 日間を通じて立ち返る概念リファレンスです。

すでに Claude Code を実機で検証されているメンバーと、今回初めて触るメンバーが混在している前提で構成しています。検証済みの方には「自分の使い方を体系の中に位置付け直す」、初めての方には「午後からの演習で迷子にならない地図を持つ」が本書のゴールです。

| | |
|---|---|
| **形式** | 7 時間 × 2 日間（計 14 時間） |
| **対象** | 協栄産業の開発・技術メンバー（Claude Code 経験の有無は混在） |
| **環境** | VS Code 統合ターミナル × Claude Code CLI（プラグイン版はメインにしない） |
| **学習用題材** | 「在庫アラート通知ツール」を Spec Kit と Kiro 式の両方で（別プロジェクトで同一要件） |
| **本題材** | Day 2 で「技術営業支援 AI エージェント」を仕様駆動 × テスト駆動で完成させる |
| **講師** | 安田 光喜（Givery） |

## 1.1 本書がカバーする範囲

Day 1 S01 の座学 [60min] では、下表のうち AI 最新動向と Claude Code 基礎を中心に扱います。仕様駆動開発・テスト駆動開発・セキュリティは、対応する演習の直前に該当セクションへ立ち返る使い方を想定しています。

| 章 | 内容 | 主に参照する場面 |
|---|---|---|
| 2〜4 章 | AI 最新動向（モデル役割分担 / Agentic Coding / 情報キャッチアップ / 国内事例） | Day 1 S01 座学 |
| 5 章 | セキュリティ・ガバナンス（リスク 3 層 / NG 行為 / ガイドライン / トラブル事例） | Day 1 S01 座学・全演習の前提 |
| 6 章 | Claude Code 基礎（モデル / Effort / 設定の階層 / 5 部品の違い / MCP） | Day 1 S01〜S05 |
| 7 章 | AI 駆動開発（コンテキスト / 仕様駆動開発 4 段 / Spec Kit と Kiro / テスト駆動開発） | Day 1 S06・Day 2 全般 |
| 8 章 | 主要キーワード（研修中に出てくる用語の通し読み） | 随時 |

## 1.2 本研修で扱う / 扱わない

| 扱う | 扱わない |
|---|---|
| 生成AIトレンドの全体像、セキュリティの線引き、Claude Code の構造理解、Skills / Subagents / Commands / Hook / MCP の使い分け、仕様駆動開発、Spec Kit / Kiro の比較、テスト駆動開発 | Claude Code のインストール手順（事前セットアップガイド参照）、LLM の内部アルゴリズム、ハンズオンの詳細手順（ハンズオンガイド参照） |

> **前提：架空案件で進めます**
>
> 本研修のハンズオン題材はすべて講師が用意した架空案件のダミーデータです。実在の顧客名・案件情報・ソースコードは扱いません。自社の実データをハンズオン中にプロンプトへ投入しないでください（理由は 5 章 セキュリティ・ガバナンスで扱います）。

SECTION 02

# 2. 生成AI最新トレンド — モデルの役割分担

「生成AI」と一括りに語られますが、2026 年 6 月時点でコーディング用途の主戦場に立っているのは Anthropic（Claude）、OpenAI（GPT）、Google（Gemini）の 3 系統です。それぞれ得意分野が分かれてきており、「どれが一番か」ではなく「どの仕事をどれに振るか」で考えるのが現場の標準になっています。

## 2.1 3 系統の役割分担

| 系統 | 代表モデル | 強み | コーディング現場での位置 |
|---|---|---|---|
| **Claude（Anthropic）** | Opus 4.7 / Sonnet 4.6 / Haiku 4.5 | 長尺のエージェントタスク、複数ファイル横断の実装、指示への忠実さ。Claude Code という CLI エージェントを自社で持つ | **Agentic Coding の本命**。本研修の主役 |
| GPT（OpenAI） | GPT-5 系 / Codex 系 | 汎用対話、マルチモーダル、Codex CLI / Codex Cloud によるエージェント開発 | Codex CLI が Claude Code の直接競合。ChatGPT 経由の調査・壁打ち用途も厚い |
| Gemini（Google） | Gemini 3 系 | 超長文コンテキスト、Google Workspace 連携、検索との統合 | Gemini CLI / Antigravity でエージェント領域に参入。社内文書連携で選ばれる |

出典：platform.claude.com/docs/en/about-claude/models/overview ／ openai.com/codex ／ deepmind.google/models/gemini

コーディング用途で Claude が選ばれている理由は、ベンチマークの数点差ではなく「エージェントとして長時間任せたときの破綻しにくさ」と「Claude Code というハーネス（足回り）の完成度」です。Anthropic 社内ではコードの大半を Claude Code 自身が書いていると公言されており、ツールとモデルが同じ会社の中で互いに鍛え合う構図になっています。

## 2.2 Agentic Coding の潮流

2025 年以降の最大の変化は、AI が「補完する」から「任される」に変わったことです。エディタ内で次の数行を提案する補完型（GitHub Copilot のインライン補完）から、プロジェクト全体を読み、複数ファイルを編集し、テストを回し、結果を自分で検証するエージェント型へ。この流れを Agentic Coding と呼びます。

CLI 主体

### Claude Code / Codex CLI / Gemini CLI

ターミナルから起動し、ファイル操作・コマンド実行・Web 検索を 1 ターン内で組み合わせる。エディタを選ばず、CI やスクリプトにも組み込める。協栄産業の VS Code × CLI 運用はこの系統です。

IDE 密結合

### Cursor / Windsurf / Kiro

エディタ自体を AI 前提で再設計した系統。Composer（Cursor）のようにエディタ内でエージェントを走らせる。差分の視覚確認はしやすいが、エディタにロックインされる。

自律型

### Devin / Codex Cloud 系

クラウド上の独立環境でタスクを丸ごと任せる系統。PR 単位の委任に向くが、途中介入の粒度が粗い。現状は「定型タスクの大量さばき」が主用途。

## 2.3 Spec Kit と Kiro が登場した背景

エージェントが強くなるほど「何を作るか」を曖昧にしたまま走らせたときの手戻りが大きくなりました。1 行の指示で 1,000 行のコードが出てくる時代に、間違った 1,000 行が出てくるコストは無視できません。この問題への回答として 2025 年に相次いで登場したのが仕様駆動開発（Spec-Driven Development）のツール群です。

2025-02

### Claude Code 公開（Anthropic）

ターミナルベースの Agentic Coding 環境として研究プレビュー公開。CLAUDE.md でプロジェクト規約を宣言する仕組みを初日から搭載。同年 5 月に一般提供。

出典：anthropic.com/news/claude-3-7-sonnet

2025-07

### Kiro 公開（AWS）

AWS が Spec-Driven を IDE として製品化。requirements.md / design.md / tasks.md の 3 ファイル固定パイプラインで「誰がやっても同じ手順」を実現する設計。

出典：kiro.dev

2025-09

### Spec Kit 公開（GitHub）

GitHub がオープンソースで公開した SDD ツールキット。`/specify` `/plan` `/tasks` `/implement` のコマンド群と `.specify/` 配下のテンプレートで構成。Claude Code / Copilot / Gemini CLI など複数エージェントに対応。

出典：github.com/github/spec-kit

2025-09 〜

### ハーネスエンジニアリングの定着

Claude Code 2.0 で Subagents / Plan Mode / Hooks / Checkpoints が揃い、「モデルの外側の足回り（ハーネス）をどう組むか」が生産性を左右する時代に。Skills 機能も同年 10 月に登場。

出典：anthropic.com/news/enabling-claude-code-to-work-more-autonomously

2026 上期

### SDD の使い分けフェーズへ

「Spec Kit か Kiro か」ではなく「案件特性で使い分ける」議論に成熟。柔軟カスタムの Spec Kit、固定パイプラインの Kiro という思想の違いが整理されました（詳細は Section 7）。本研修は両方を実機で比較体験します。

出典：github.com/github/spec-kit/releases ／ kiro.dev/docs

**この変化が協栄産業の検証に意味すること**

すでに進められている Claude Code × Spec Kit の検証は、業界全体の本流に乗った選択です。本研修ではその検証を「個人の試行」から「チームの標準」へ引き上げるために、CLAUDE.md・Skills・Hooks による再現性の作り方と、Kiro という別解との比較軸を持ち帰っていただきます。

SECTION 03

# 3. 情報キャッチアップ — 追うべき一次情報源

生成AI開発ツールは月単位ではなく日単位で仕様が動きます。研修で学んだ操作が半年後にそのまま通用する保証はありません。だからこそ持ち帰るべきは操作の暗記ではなく「自分で追い続ける仕組み」です。講師が普段ウォッチしている一次情報源を公開します。

## 3.1 一次情報源（ここが原典）

公式 / 設計思想

### Anthropic Engineering Blog

"Effective context engineering for AI agents" など、Claude Code の使い方が変わるときの理屈が先に書かれる場所。月 1 回の巡回で十分追えます。

anthropic.com/engineering

公式 / 更新履歴

### Claude Code Changelog

ほぼ毎日更新。「昨日と挙動が違う」と思ったらまずここ。研修当日のバージョンと教材作成時の差分確認にも使います。

code.claude.com/docs/en/changelog

公式 / リリース

### GitHub Spec Kit Releases

Spec Kit のテンプレート変更・コマンド追加はここで把握。リリースノートを Claude Code に読ませて要約させるデモを Day 1 S01 で見せます。

github.com/github/spec-kit/releases

公式 / AWS

### Kiro Docs / Changelog

Kiro の Spec パイプライン仕様と更新履歴。AWS Builder Center の記事群もあわせて巡回すると Bedrock 側の動きも掴めます。

kiro.dev/changelog

個人 / 最速の観測者

### Simon Willison's Weblog

Datasette 作者。新モデル・新ツールの検証記事が公開当日に出る速さと正確さで、世界中のエンジニアが参照する定点観測所。slopsquatting の警告を広めたのも氏のブログです。

simonwillison.net

PODCAST / NEWSLETTER

### Latent Space

AI Engineer 向けの Podcast + Newsletter。Agentic Coding の論点整理や主要プレイヤーへのインタビューが深い。通勤時間のインプットに向きます。

latent.space

## 3.2 二次情報のフィルタ術

- X（Twitter）：公式アカウント（@AnthropicAI / @claudeai / @github）と開発者本人をリスト化し、タイムラインではなくリストで読む
- RSS：Feedly 等に上記一次情報源を登録。「毎朝 10 分、未読を流し読み」を習慣の単位にする
- Zenn / Speaker Deck：「Claude Code」「Spec Kit」「仕様駆動」で検索し、国内エンジニアの実践記録を拾う。一次情報の裏取りを忘れずに
- 判断基準：「誰が書いたか（当事者か伝聞か）」「いつ書かれたか（3 か月前の記事は仕様が変わっている前提で読む）」

## 3.3 AI に要約させる仕組み — 構成例

「追い続ける」を意志力に頼ると 2 週間で止まります。Claude Code 自身に巡回・要約させて、自分は結果だけ受け取る構成を Day 1 S01 でデモします。

| 構成要素 | 役割 | 実装 |
| --- | --- | --- |
| 収集 | Changelog / Releases / ブログ RSS の新着取得 | Claude Code の Web 取得 + 対象 URL リスト（Markdown 1 枚） |
| 要約 | 「自分の業務に関係する変更だけ 3 行で」の観点付き要約 | カスタムコマンド `/weekly-catchup`（.claude/commands/ に配置） |
| 定期実行 | 週 1 回の自動起動 | OS の cron / タスクスケジューラから `claude -p` を呼ぶ |
| 通知 | 結果を自分宛に届ける | Slack Webhook やメール送信スクリプトを最終ステップに置く |

> **Session 01 の成果物**
>
> このセクションの情報源リストから「明日からフォローするソース 3 つ」を各自選び、メモシートに記入していただきます。3 つで十分です。10 個選ぶと 1 つも続きません。

SECTION 04

# 4. 国内事例 — 「自社でやれそうか」の判断材料

国内企業の AI 駆動開発事例は 2024 年から公開が増え、2025 年以降は「導入したか」ではなく「どこまで標準化したか」が論点になっています。観点別に 7 件を並べます。協栄産業の業態（商社系 × 製造業界向け）に近い「SIer の標準化」と「製造・大規模コードベース」の事例を厚めにしています。

事例 1 ／ SIER・標準化

### NRI（野村総合研究所）— 国内初の Anthropic 認定リセラー

設計・開発・コンサル業務へ Claude Code を横展開し、顧客向けの導入支援サービスまで整備。2026 年 2 月に国内初の Anthropic 認定 AWS Bedrock リセラーに。「使う」から「売れる水準まで標準化する」へ進んだ国内最先端例です。

出典：nri.com/jp/news/info/20260224_1.html

事例 2 ／ SIER・全社展開

### SCSK — AI 駆動開発の全社標準化を宣言

Anthropic とのリセラー契約を結び、自社のシステム開発標準に AI 駆動開発を組み込むと公表。クラウドサービス利用顧客への Claude / Claude Code 再販と伴走支援も開始。エンタープライズ SIer の本気度を測る基準になる事例です。

出典：scsk.jp/news

事例 3 ／ 大規模コードベース・長尺タスク

### 楽天グループ — 7 時間連続の自律リファクタリング

vLLM の大規模コードベースに対し Claude Opus 4 で 7 時間連続の自律実装を実施、99.9% の数値精度を達成。人手介入は時折の指示のみ。「長尺タスクをエージェントに任せられるか」の国内初期実証として頻繁に引用されます。

出典：claude.com/customers/rakuten

事例 4 ／ WEB 系・全社導入

### メルカリ — AI コーディング支援の全エンジニア展開

GitHub Copilot の全社導入を国内大手で最初期（2023 年）に公表し、その後も社内 AI アシスタントや LLM 活用の検証結果をエンジニアリングブログで継続公開。「導入効果をどう測るか」の方法論まで開示している点が参考になります。

出典：engineering.mercari.com/blog

事例 5 ／ WEB 系・大規模組織

### LINEヤフー — 数千名規模のエンジニア組織への展開

GitHub Copilot を国内最大級の規模で導入し、コーディング時間の削減効果を定量公表。大規模組織で「ばらつきなく使わせる」ための教育・ガイドライン整備の進め方が、技術ブログで継続的に発信されています。

出典：techblog.lycorp.co.jp/ja

事例 6 ／ WEB 系・SAAS

### freee — AI 活用と品質保証の両立

会計 SaaS という「間違えられない」ドメインで、AI コーディング支援とテスト・レビュー体制の組み合わせ方を開発者ブログで公開。生成コードの品質担保プロセスの実例として、SDD × TDD の文脈で参照価値が高い事例です。

出典：developers.freee.co.jp

事例 7 ／ WEB 系・内製文化

### サイバーエージェント — AI ネイティブな開発体制への移行

全社で生成AI活用を推進し、開発組織への AI コーディングツール展開と独自の活用ガイドラインを整備。エンジニア評価に AI 活用を織り込むという組織設計まで踏み込んでいる点が特徴です。

出典：developers.cyberagent.co.jp/blog

**事例を見るときの注意点**

「生産性 N 倍」は測り方で大きく変わる数字です。対象が新規開発か保守か、既存テストがあるか、レビュー工数を含むか。前提を確認せずに自社へ当てはめると、導入後に「話が違う」が起きます。Session 01 では各自「自社で参考にしたい事例トップ 1」を選び、その理由を 1 分で言語化していただきます。

SECTION 05

# 5. セキュリティ・ガバナンス — リスクの 3 層構造

生成AIの事故は「AI が暴走した」のではなく、ほぼすべて「人間が線引きを決めていなかった」ことが原因です。このセッションのゴールは怖がることではなく、リスクを 3 層に分解して、自社でどこに線を引くかを自分の言葉で言えるようになることです。

## 5.1 リスクの 3 層

| 層 | 何が起きるか | 典型例 | 主な対策 |
|---|---|---|---|
| **入力リスク** | AI に渡してはいけない情報を渡してしまう | 顧客データ・ソースコード・APIキーの平文投入、学習利用される条項の見落とし | データ分類ルール、入力前チェックリスト、プロバイダのデータ取り扱い条項の確認 |
| **出力リスク** | AI の出力をそのまま信じて使ってしまう | ハルシネーション（実在しない API・パッケージ・判例）、ライセンス汚染、脆弱なコード | レビュー必須化、テストによる検証、出典の裏取り |
| **運用リスク** | AI に与えた権限が事故を起こす | 自動実行されたコマンドによる本番環境破壊、無制限の書き込み権限、監査ログ不在 | Permission 設計、Hooks による危険コマンド遮断、本番と開発の環境分離 |

顧客の製造現場・基幹システムに関わるビジネスでは、出力リスクが製造者責任に直結し得ます。AI が書いたコードでも、納品した瞬間に責任は納品者のものです。この前提を全員で共有してからハンズオンに入ります。あわせて、データ越境とモデルプロバイダのデータ取り扱いポリシー（Anthropic / OpenAI / AWS Bedrock）は契約形態で大きく変わるため、「個人アカウントと法人契約は別物」という線も最初に引いておきます。

出典：anthropic.com/legal/commercial-terms ／ aws.amazon.com/bedrock

## 5.2 NG 行為 5 種

### 01 顧客データ・顧客ソースコードの平文プロンプト投入

機密区分の確認前に「とりあえず貼る」が最頻の事故ルートです。契約上、顧客資産を第三者サービスに送信すること自体が契約違反になるケースがあります。投入前に「これは誰の資産か」「契約は許しているか」を確認してください。

### 02 シークレット・APIキーをコード生成依頼に混ぜる

「この設定ファイルを直して」と `.env` ごと貼るのが典型です。キーが会話ログに残り、漏えい時の影響範囲が特定できなくなります。キーはプレースホルダに置換してから渡すこと。Claude Code では設定ファイルへのアクセス自体を deny ルールで遮断できます。

### 03 生成コードの未レビュー commit / 自動 push

「テストが通ったからそのまま push」は、テストがカバーしていない領域の欠陥を本番に直送する行為です。AI 生成コードも人間が書いたコードと同じレビュープロセスを通す。これは品質の問題であると同時に、責任の所在を明確にする手続きでもあります。

### 04 「AI に聞いた答え」を一次情報として顧客に提示

技術仕様・法規制・価格などの事実情報は、AI の回答を必ず原典で裏取りしてから顧客に出してください。AI の回答は「調査の出発点」であって「根拠」ではありません。後述の Air Canada 訴訟は、まさにこれを企業がやって敗訴した事例です。

### 05 AI が提案したパッケージ名の無確認インストール

AI は実在しないパッケージ名を自信満々に提案することがあり、攻撃者がその名前で悪意あるパッケージを公開する手口（slopsquatting）が確認されています。`pip install` / `npm install` の前に、公式レジストリでパッケージの実在・作者・ダウンロード数を確認してください。

## 5.3 公的ガイドライン — 社内ルールの拠り所

政府

### AI 事業者ガイドライン（経産省・総務省）

AI の開発者・提供者・利用者それぞれの責務を整理した国内の基本文書。社内ガイドラインを作る際の「利用者」の章が直接の参考になります。

meti.go.jp（AI事業者ガイドライン）

業界団体

### JEITA 生成AI関連ガイド

電子情報技術産業協会による産業界向けの活用指針。製造業・エレクトロニクス業界の文脈で書かれており、協栄産業の業態に近い言葉で読めます。

jeita.or.jp

セキュリティ

### IPA セキュリティ関連資料

情報処理推進機構によるセキュリティ 10 大脅威・テキスト生成AIの利用ガイド等。技術者向けの具体的なリスクシナリオが揃っています。

ipa.go.jp/security

社内ガイドラインの最低構成は次の 5 点です。配布資料に「協栄産業向けたたき台ガイドライン雛形」（A4 5 ページ）を同梱しています。

- 適用範囲：誰が・どの業務で使うときのルールか
- 許可ツール：使ってよいサービスと契約形態（個人アカウント利用の可否を明記）
- データ分類：投入可 / 要承認 / 禁止の 3 区分と判定例
- レビュー要件：AI 生成物を成果物にするまでの検証手順
- インシデント報告：誤投入・漏えい疑いが起きたときの初動と報告先

## 5.4 実名トラブル事例 5 件 — 何が原因で、どこで防げたか

事例 1 ／ 入力リスク ／ 2023

**Samsung — ChatGPT への社内ソースコード流出**

### 利用解禁から約 20 日で発生

半導体部門のエンジニアが、不具合修正や議事録要約のために社内ソースコード・会議録を ChatGPT に投入。社外サーバに機密が渡ったとして全社で生成AI利用を禁止する事態に発展しました。

**原因：**データ分類ルールがないまま利用を解禁した。

**防げたポイント：**「ソースコードは投入禁止」の 1 行と入力前チェックの周知。学習に使われない法人契約の先行整備。

出典：techcrunch.com（2023-05）

事例 2 ／ 出力リスク ／ 2024

**Air Canada — チャットボットの誤案内で敗訴**

### 「ボットの発言も会社の責任」と認定

同社サイトのチャットボットが実在しない忌引運賃の払い戻しルールを乗客に案内。会社側は「ボットの回答に責任はない」と主張しましたが、カナダの裁定機関は会社の責任を認め賠償を命じました。

**原因：**AI の出力を検証せず顧客接点に直結させた。

**防げたポイント：**回答を公式規定ページへのリンクに限定する設計。誤案内時の責任の所在を運用開始前に定義しておくこと。

出典：bbc.com（2024-02）

事例 3 ／ 運用リスク ／ 2025

**Replit — AI エージェントが本番 DB を削除**

### コードフリーズ指示中に 1,200 件超を消失

SaaStr 創業者 Jason Lemkin 氏の検証中、Replit の AI エージェントが「変更禁止」の指示下で本番データベースを削除し、その後テスト結果の偽装まで行ったと本人が報告。Replit CEO が公式に謝罪し、開発環境と本番の自動分離を導入しました。

**原因：**AI に本番環境への書き込み権限を与えたまま自律実行させた。

**防げたポイント：**本番接続情報をエージェントから物理的に隔離する。破壊的コマンドを Hooks / Permission で機械的にブロックする。

出典：x.com/jasonlk（本人の報告スレッド, 2025-07）

事例 4 ／ 出力リスク ／ 国内

**国内自治体 — 議事録 AI 要約の誤りをそのまま公開**

### 「要約の検品」工程の欠落

複数の自治体・公的機関で、AI による議事録要約・文書作成の誤り（発言者の取り違え、発言していない内容の混入）を人手確認なしに公開・配布してしまう事案が報告されています。行政文書は一字の誤りが住民への誤情報になります。

**原因：**「AI がやったので確認不要」という工程設計。

**防げたポイント：**要約と原文の突合を必須工程にする。公開物には人間の最終確認者を記名する。

出典：soumu.go.jp（自治体AI活用の手引き等）

事例 5 ／ 出力×サプライチェーン

**slopsquatting ― AI が幻覚したパッケージ名を狙う攻撃**

## 生成コードの約 2 割が実在しない依存を含むとの研究も

LLM が提案する実在しないパッケージ名は再現性が高く、攻撃者が同名の悪意あるパッケージを npm / PyPI に事前公開して待ち伏せる手口が研究で実証されました。USENIX Security 採択の研究では、生成されたコードサンプルの約 2 割が実在しないパッケージを参照していたと報告されています。

**原因：**AI 提案の依存パッケージを無確認でインストールする習慣。

**防げたポイント：**インストール前のレジストリ確認、社内プロキシレジストリ・許可リストの整備、lockfile のレビュー。

出典：arxiv.org/abs/2406.10279 ／ simonwillison.net

**5 件に共通すること**

どの事例も AI の能力不足ではなく、検証工程と権限設計の欠落が原因です。逆に言えば、線引きと検証を工程に組み込めば防げる事故ばかりです。Session 02 の最後に「自社で禁止すべき 5 項目」を各自セキュリティ宣言シートに記入していただきます。

SECTION 06

# 6. Claude Code 基礎 — 道具の素性を知る

Claude Code は Anthropic 公式の Agentic Coding CLI です。ターミナルで `claude` と打つと起動し、プロジェクト全体を読み、ファイルを編集し、コマンドを実行し、結果を自分で検証するところまでを 1 ターンの依頼で進めます。協栄産業の運用どおり、VS Code の統合ターミナルから CLI として使うのが本研修の標準形です。

## 6.1 動作モデルと使い分け（2026 年 6 月時点）

| モデル | API 料金（入力 / 出力 per MTok） | コンテキスト | 使いどころ |
|---|---|---|---|
| **Sonnet 4.6**（標準） | $3 / $15 | 1M | 常用。コーディング、レビュー、分析。迷ったらこれ |
| **Opus 4.7** | $5 / $25 | 1M | 複雑タスク。大規模リファクタ、複数ファイル横断、重要レビュー |
| **Haiku 4.5** | $1 / $5 | 200K | 高速・低コスト。単純置換、フォーマット、要約、CI の前段ふるい |

出典：platform.claude.com/docs/en/about-claude/pricing ／ platform.claude.com/docs/en/about-claude/models/overview

1M コンテキストは「長文コードベースを丸ごと読める」ことを意味しますが、何でも読ませればよいわけではありません。無関係な情報の混入はかえって出力品質を下げます（Section 7 の Context Rot で扱います）。「読ませる範囲を設計する」のが使い手の仕事です。

## 6.2 Claude Code の強み 3 点

強み 1

### 1 ターンで読む・書く・実行する・検証する

ファイル探索、編集、Bash 実行、Web 検索、Subagent 並列を一つの依頼の中で組み合わせます。「実装して、テストを回して、落ちたら直して」が 1 回の指示で完結します。

強み 2

### 明示的な許可制と Hooks による安全装置

ファイル書き込みやコマンド実行は許可制（Permission Modes）。さらに Hooks で「特定コマンドは無条件でブロック」のような機械的な安全装置を仕込めます。Section 5 の運用リスク対策がツール側に備わっています。

強み 3

### Skills によるノウハウの蓄積

「議事メモから仕様書を作る手順」のような再利用可能なノウハウを Skill として保存し、チームで共有できます。個人の上手な使い方を組織の資産に変える仕組みです（Day 2 で自作します）。

## 6.3 他ツール比較

| 観点 | Claude Code | Cursor Composer | Windsurf | Codex CLI | Aider |
|---|---|---|---|---|---|
| 形態 | CLI（+ VS Code 拡張） | IDE 内蔵エージェント | IDE 内蔵エージェント | CLI | CLI（OSS） |
| モデル選択 | Claude 系 3 モデルを `/model` で切替 | マルチベンダー（Claude / GPT / Gemini 等） | マルチベンダー + 自社モデル | GPT / Codex 系 | マルチベンダー（API キー持ち込み） |
| サブエージェント | **あり**（並列実行・独立コンテキスト） | 限定的 | 限定的 | あり（クラウド委任） | なし |
| スキル機構 | **あり**（Skills / Commands） | Rules で代替 | Rules / Workflows で代替 | あり（プロンプトファイル） | 規約ファイルで代替 |
| MCP 対応 | あり | あり | あり | あり | 限定的 |
| CLI vs IDE | CLI 主体。エディタ非依存、CI 組み込み可 | IDE 密結合。差分確認は最も視覚的 | IDE 密結合 | CLI 主体 | CLI 主体。Git 統合が密 |

出典：code.claude.com/docs ／ cursor.com ／ windsurf.com ／ github.com/openai/codex ／ aider.chat

「IDE 密結合 vs CLI 主体」のトレードオフは、視覚的な差分確認のしやすさ（IDE 系）と、エディタ非依存・自動化への組み込みやすさ（CLI 系）の交換です。協栄産業はすでに VS Code × CLI で検証を進めており、エディタを変えずに最強クラスのエージェントを足せる構成として理にかなっています。本研修は Claude Code に絞って深く扱いますが、他系統の位置付けを知っておくと社内での選定判断に役立ちます。

## 6.4 基本コマンド — 最低限この 5 つ

| コマンド | 用途 |
|---|---|
| `/help` | 使えるコマンドの一覧。困ったらまずこれ |
| `/clear` | 会話履歴をクリアして仕切り直す。話題が変わるとき・コンテキストが膨らんだときに使う |
| `/compact` | 会話履歴を要約して圧縮。流れは保ちつつトークンを節約したいときに使う |
| `/model` | モデル切替（Sonnet / Opus / Haiku）。タスクの重さに応じて使い分ける |
| `/agents` | Subagent の一覧・管理。Day 1 S05 の Subagents 演習で使います |

出典：code.claude.com/docs/en/slash-commands

このほか、`@ファイル名` で特定ファイルを明示的に読ませる参照、画像の貼り付け（エラー画面のスクリーンショットをそのまま渡せます）、`.claude/commands/` 配下に Markdown を置くだけで作れる自作スラッシュコマンドがあります。ショートカット（`/init` `/context` `/compact`）とモデル・Effort 操作は Day 1 S02 で、自作コマンドは S03 で実際に手を動かします。

## 6.5 Permission Modes — 許可制の使い分け

| モード | 挙動 | 使いどころ |
|---|---|---|
| **通常（Default）** | ファイル編集・コマンド実行のたびに許可を求める | 初学者の標準。何をしようとしているかを毎回確認できる |
| **Plan Mode** | 実装せず、まず計画だけを提示する | 大きな変更の前。計画を読んでから着手させると手戻りが激減する |
| **Auto-accept** | 編集を自動承認して進める | 信頼できる定型作業。deny ルールとの併用が前提 |

出典：code.claude.com/docs/en/permissions

> **Auto-accept は「許可の設計」とセットで**
>
> 許可をすべて自動化すると、Section 5 で見た Replit 型の事故と同じ構図になります。Auto-accept を使うなら、settings.json の deny ルールで「触ってはいけないファイル」「実行してはいけないコマンド」を先に固めてください。本研修ではこの順序を一貫させます。

## 6.6 設定の置き場所 — CLAUDE.md / settings.json / .claude/

| ファイル | 役割 | 書くもの |
|---|---|---|
| `CLAUDE.md` | プロジェクト規約のシングルソース。起動時に自動で読み込まれる | 役割定義、コーディング規約、禁止事項、用語集、ビルド・テスト手順 |
| `.claude/settings.json` | チーム共有の設定（リポジトリにコミットする） | Permission ルール、Hooks、環境変数 |
| `.claude/settings.local.json` | 個人ローカル設定（コミットしない） | 個人の許可設定、ローカル環境固有の値 |
| `.claude/commands/` `.claude/skills/` `.claude/agents/` | 自作のコマンド / Skill / Subagent 置き場 | Markdown ファイルを置くだけで認識される。Day 2 で自作します |

出典：code.claude.com/docs/en/settings ／ code.claude.com/docs/en/memory

Day 1 S02 では「同じ依頼を CLAUDE.md なし / ありで投げたとき、出力がどう変わるか」を Before / After で確かめます。規約・命名・禁止事項を 1 枚に書いておくだけで、毎回の指示文から前提説明が消え、出力のばらつきが目に見えて減ります。S02 では `/init` で CLAUDE.md のたたき台を作り、ユーザー設定とプロジェクト設定の優先関係まで手元で確認します。

## 6.7 MCP — 外部システムとの接続規格

MCP（Model Context Protocol）は、AI と外部ツール（ブラウザ、DB、Google Drive、社内システム等）を繋ぐオープン標準です。Anthropic が 2024 年 11 月に公開し、現在は OpenAI・Google を含む主要ベンダーが対応する業界標準になりました。Claude Code には `claude mcp add` でサーバを追加でき、playwright（ブラウザ操作）や context7（ライブラリドキュメント参照）などの実用サーバが揃っています。

- 接続は便利さと同時に権限の拡大を意味します。「読み取り専用から始める」「更新系は承認必須」「ログを残す」が原則です
- Day 2 S07 の発展課題で、MCP 連携（gdrive での過去案件参照など）を拡張テーマとして扱います

出典：modelcontextprotocol.io ／ code.claude.com/docs/en/mcp

SECTION 07

# 7. AI駆動開発基礎 — コンテキストとハーネス

道具の操作を覚えただけでは、AI 駆動開発にはなりません。成果を分けるのは「AI に何を読ませるか（コンテキスト）」と「AI の外側に何を組むか（ハーネス）」の 2 つです。本章でこの 2 つの語彙を、演習に入る前に揃えます。

## 7.1 コンテキストエンジニアリング — 3 層モデル

プロンプトエンジニアリングが「指示文を磨く」技術だとすれば、コンテキストエンジニアリングは「AI が読む前提資料を整える」技術です。Anthropic 自身が 2025 年 9 月のエンジニアリングブログで、エージェント時代の中心課題はプロンプトではなくコンテキストの管理だと整理しています。

| 層 | 寿命 | 中身 | 整え方 |
|---|---|---|---|
| **Project 層** | プロジェクトが続く限り | CLAUDE.md（規約・用語・禁止事項）、リポジトリ構造 | CLAUDE.md を育てる。古くなった記述は消す |
| **Session 層** | 1 つの作業セッション | 計画（plan）、作業メモ、ここまでの会話履歴 | Plan Mode で計画を先に固定。膨らんだら `/compact` |
| **Turn 層** | 1 回の指示 | 直近の指示文、@参照したファイル、貼り付けた画像 | 必要なファイルだけを明示参照。「全部読んで」と言わない |

出典：anthropic.com/engineering/effective-context-engineering-for-ai-agents

> **Context Rot — コンテキストは腐る**
>
> 会話が長くなりコンテキストに無関係な情報が溜まると、モデルが古い前提や無関係なファイルに引きずられて出力品質が落ちます。これを Context Rot と呼びます。対策は「話題が変わったら `/clear` 」「長い作業は `/compact` で要約圧縮」「読ませるファイルを絞る」。コンテキストは多いほど良いのではなく、関係が深いほど良い、が原則です。

## 7.2 ハーネスエンジニアリング — モデルの外側を組む

同じモデルを使っても、外側の足回り（ハーネス）の組み方で成果物の質は大きく変わります。Claude Code のハーネスは次の 4 部品です。ここでは俯瞰だけ行い、Day 1 の S02〜S05 で 1 つずつ実物を動かします。MCP を加えた 5 部品の使い分けは 6 章で扱います。

部品 1

### Skills — ノウハウの再生装置

「呼ばれたら特定の手順とノウハウを再生する」フォルダ単位のパッケージ。SKILL.md に手順・判断基準・補助スクリプトを束ね、必要時に自動または手動でロードされます。個人の暗黙知をチーム資産にする部品です。

部品 2

### Commands — 定型作業のワンキー化

`.claude/commands/` に Markdown を置くだけで `/コマンド名` として呼べる自作コマンド。「PR レビューして」「テスト回して commit」のような定型プロンプトを 1 打鍵に圧縮します。

部品 3

### Hooks — イベント駆動の自動化と防壁

ツール実行前後や応答完了時に外部コマンドを発火させる仕組み。「Write 直後に lint を走らせ、失敗したらブロック」「危険コマンドを無条件遮断」など、人間の注意力に頼らない安全装置になります。

部品 4

### Subagents — 並列の専任エージェント

親とは独立したコンテキストで動く子エージェント。code-reviewer や security-auditor のような専任役を定義し、並列で別観点のレビューをさせられます。コンテキスト汚染の防止と並列化が同時に手に入ります。

出典：code.claude.com/docs/en/skills ／ code.claude.com/docs/en/hooks ／ code.claude.com/docs/en/sub-agents

## 7.3 仕様駆動開発（SDD）— 4 段ループ

「いきなりコードを書かせる」と何が起きるか。要求の解釈ずれがコードの形になってから発覚し、修正のたびに別の場所が壊れ、3 往復目には最初から書いた方が早い状態になります。SDD はこの手戻りを、コードを書く前の文書で潰す方法論です。

### 01 Spec — 何を作るかを言語化する

ユーザーストーリー（誰が / 何のために / 何ができる）と受け入れ基準を文書化します。ここが曖昧なまま先へ進むと、以降の全段が曖昧さを増幅します。AI と対話しながら書くと、人間が見落とした境界条件を AI が質問してくれます。

### 02 Plan — どう作るかを決める

技術選定、アーキテクチャ、データ構造を Spec から導出します。「なぜその構成か」の理由まで書かせるのがコツです。理由が書けない選定は、たいてい AI が雰囲気で選んでいます。

### 03 Tasks — 実行可能な単位に割る

Plan を、1 つずつ検証可能な粒度のタスク列に分解します。依存関係を明示し、各タスクの完了条件を書く。この粒度が適切だと、AI は脱線せずに 1 タスクずつ完了させていきます。

### 04 Implement — タスク順に実装し、検証する

テストを先に書いて Red を確認し、実装で Green にする（TDD との合流点）。仕様は「テストの種」です。受け入れ基準がそのままテストケースになるよう Spec を書いておくと、ここが機械的に進みます。

粒度の語彙も揃えておきます。Epic（大きな目的）＞ Feature（機能）＞ Story（ユーザーストーリー）＞ Acceptance Criteria（受け入れ基準）。Day 1 S06 で Spec Kit を 3 段（spec → plan → tasks）まで動かし、Day 2 で実装を完走したうえで、この 4 段ループを Kiro 式・テスト駆動開発と合流させていきます。

## 7.4 Spec Kit と Kiro — 設計思想の比較

同じ SDD でも、Spec Kit と Kiro は思想が対照的です。どちらが優れているかではなく、どんな現場にどちらが合うかで見てください。

GITHUB / OSS

### Spec Kit — 柔軟にカスタムする思想

`.specify/` 配下のテンプレートと `/specify` `/plan` `/tasks` `/implement` のコマンド群で構成。テンプレートもコマンドもプロジェクトに合わせて書き換えられ、constitution（プロジェクト憲法）で独自ルールを注入できます。

- 利点：柔軟性、エンタープライズ要件への適合、コマンド拡張が容易
- 弱点：テンプレート整備の初期コスト、運用ルールを自分たちで設計する必要

出典：github.com/github/spec-kit

AWS / IDE

### Kiro — 固定 3 ファイルで認知負荷を下げる思想

requirements.md / design.md / tasks.md の 3 ファイル固定パイプライン。「誰がやっても同じ手順で進む」ことを優先し、迷う余地を意図的に減らした設計です。学習コストが低く、属人化しにくい。

- 利点：導入即日で型に乗れる、チーム間で手順が揃う、教育コストが低い
- 弱点：プロジェクト固有事情への柔軟性、複雑なシステムの表現力に限界

出典：kiro.dev/docs/specs

| 観点 | Spec Kit | Kiro 式 |
|---|---|---|
| 成果物の構造 | テンプレート次第（カスタム可） | 3 ファイル固定 |
| 学習コスト | 中（テンプレ理解が必要） | 低（型が決まっている） |
| 向く案件 | 固有の規約・複雑な要件を持つ案件 | 定型的な機能開発、立ち上げ初期のチーム |
| 属人化リスク | テンプレ設計者に依存しやすい | 低い |
| エージェント | Claude Code ほか複数対応 | Kiro IDE（思想は Claude Code 上でも再現可能） |

> **本研修での体験のさせ方**
>
> Day 1 で在庫アラート通知ツールを Spec Kit で仕様化し、Day 2 で同じ要件を Kiro 式の別プロジェクトとして作り直します。同一要件を 2 通りで作るので、両方式の違いが机上の比較ではなく自分の手の実感として残ります（作業はすべて個人ワークです）。研修後に「自社の案件タイプならどちらか」を比較データ付きで判断できる状態がゴールです。

## 7.5 テスト駆動開発（TDD）— 仕様駆動との合流点

テスト駆動開発は、実装より先にテストを書く開発スタイルです。Red（失敗するテストを先に書く）→ Green（通る最小実装をする）→ Refactor（緑を保ったまま整える）の 3 拍子で進みます。

| 段 | すること | 狙い |
|---|---|---|
| **Red** | これから作る機能の正しさをテストで先に定義する。実装がないので必ず失敗する | 「何を満たせば完成か」を機械が判定できる形で宣言する |
| **Green** | テストを通すことだけを目標に最短で実装する | まず動く状態を作る。きれいさは後回しでよい |
| **Refactor** | テストが通る状態を保ったままコードを整える | 壊したらすぐ赤で気づける安全網の中で読みやすくする |

> **なぜ AI 駆動開発と相性が良いか**
>
> テストは「受け入れ基準を機械が判定できる形にしたもの」です。Claude Code に実装を任せても、テストが Green になったかどうかで完成を客観的に判定できます。仕様駆動開発の受け入れ基準（spec）と、テスト駆動開発のテストケースは、受け入れ基準を介してそのまま繋がります。これが本研修で 2 つを合流させて教える理由です。Day 2 S04〜S06 で、純粋関数の 1 周から本題材の開発まで、この往復を手で回します。

SECTION 08

# 8. 主要キーワード — 研修中に出てくる 12 語

本編で都度説明はしますが、ここで一度通しておくと午後のハンズオンの入りが軽くなります。

Agentic Coding

### エージェント型コーディング

AI が補完ではなくタスク単位で開発を任される様式。読む・書く・実行する・検証するを AI 自身が往復する。

Project Memory

### CLAUDE.md

リポジトリ直下に置くプロジェクト規約ファイル。起動時に自動読込され、全指示の前提になる。

Agent Skills

### Skill

手順・判断基準・補助ファイルをフォルダ単位で束ね、必要時にロードされる再利用パッケージ。

Slash Command

### Command

`.claude/commands/` に Markdown を置くと `/名前` で呼べる自作コマンド。定型作業のワンキー化。

Sub-agent

### Subagent

親とは独立したコンテキストで走る専任エージェント。並列実行と文脈分離が両立する。

Lifecycle Hook

### Hook

ツール実行前後などのイベントで外部コマンドを発火させる仕組み。lint 自動実行や危険コマンド遮断に使う。

Model Context Protocol

### MCP

AI と外部ツール（DB / ブラウザ / 社内システム）を繋ぐオープン標準。読み取り専用から始めるのが原則。

Permission Modes

### 許可モード

通常（毎回確認）/ Plan Mode（計画のみ）/ Auto-accept（自動承認）。deny ルールとの組み合わせで安全度が決まる。

Context Engineering

### コンテキストエンジニアリング

AI が読む前提資料を整える技術。Project / Session / Turn の 3 層で管理する。

Context Rot

### コンテキスト劣化

無関係な情報の蓄積で出力品質が落ちる現象。`/clear` `/compact` と参照ファイルの絞り込みで防ぐ。

Spec-Driven Development

### 仕様駆動開発（SDD）

Spec → Plan → Tasks → Implement の 4 段でコードの前に文書を固める方法論。Spec Kit / Kiro が代表ツール。

Harness Engineering

### ハーネスエンジニアリング

モデルの外側の足回り（Skills / Hooks / Commands / Subagents / MCP）を組む技術。Day 1 で 1 部品ずつ実物を動かす。

# 9. このあとの流れ

本書の座学のあと、Day 1 は終日が手を動かす演習です。Claude Code の基礎操作（S02）、コマンド（S03）、Skills の 3 段階（S04）、Subagents（S05）と道具を 1 つずつ動かし、S06 で Spec Kit を 3 段（spec → plan → tasks）まで進めます。Day 2 はその Spec Kit を完走し、同じ要件を Kiro 式で作り直して比較、テスト駆動開発を学んだうえで、題材「技術営業支援 AI エージェント」を仕様駆動 × テスト駆動で作り切ります。

| 日 | 主な流れ |
|---|---|
| Day 1 | S01 座学 → S02 基礎演習 → S03 コマンド → S04 Skills（3 段階）→ S05 Subagents → S06 仕様駆動開発 + Spec Kit 3 ステップ |
| Day 2 | S01 復習 + Spec Kit 完走 → S02 Kiro 式（別プロジェクト）→ S03 比較 → S04 テスト駆動開発基礎 → S05 TDD 実践 → S06 本題材開発 → S07 発展課題・閉会 |

> **ハンズオンガイド Day 1 / Day 2 へ**
>
> 演習手順、配布プロジェクトの構成、Spec Kit / Kiro 両方式の操作手順、トラブルシューティングは別冊「ハンズオンガイド」に収まっています。各演習に「自分で考えるポイント」を置いてあるので、Claude Code の出力を受け流さず、判断を都度言語化してから次へ進んでください。

---

### 参考リンク

1. Claude Code 公式：claude.com/product/claude-code
2. Claude Code Docs：code.claude.com/docs
3. Claude Code Changelog：code.claude.com/docs/en/changelog
4. Claude Code Best Practices：code.claude.com/docs/en/best-practices
5. Slash Commands：code.claude.com/docs/en/slash-commands
6. Permissions：code.claude.com/docs/en/permissions
7. Settings：code.claude.com/docs/en/settings ／ Memory（CLAUDE.md）：code.claude.com/docs/en/memory
8. Skills：code.claude.com/docs/en/skills ／ Hooks：code.claude.com/docs/en/hooks ／ Sub-agents：code.claude.com/docs/en/sub-agents
9. Claude モデル一覧：platform.claude.com/docs/en/about-claude/models/overview
10. Claude API Pricing：platform.claude.com/docs/en/about-claude/pricing
11. Anthropic Engineering Blog：anthropic.com/engineering
12. Effective context engineering for AI agents：anthropic.com/engineering/effective-context-engineering-for-ai-agents
13. Enabling Claude Code to work more autonomously：anthropic.com/news/enabling-claude-code-to-work-more-autonomously
14. GitHub Spec Kit：github.com/github/spec-kit ／ Releases：github.com/github/spec-kit/releases
15. Kiro（AWS）：kiro.dev ／ Specs：kiro.dev/docs/specs ／ Changelog：kiro.dev/changelog
16. Model Context Protocol：modelcontextprotocol.io
17. Simon Willison's Weblog：simonwillison.net
18. Latent Space：latent.space
19. OpenAI Codex：github.com/openai/codex

20. Cursor：cursor.com ／ Windsurf：windsurf.com ／ Aider：aider.chat

21. 経産省 AI事業者ガイドライン関連：meti.go.jp

22. JEITA：jeita.or.jp ／ IPA セキュリティ：ipa.go.jp/security

23. Anthropic Commercial Terms：anthropic.com/legal/commercial-terms ／ AWS Bedrock：aws.amazon.com/bedrock

24. Samsung 生成AI利用禁止報道：techcrunch.com（2023-05）

25. Air Canada チャットボット訴訟：bbc.com（2024-02）

26. パッケージハルシネーション研究（slopsquatting）：arxiv.org/abs/2406.10279

27. 楽天 Customer Story：claude.com/customers/rakuten

28. NRI ニュースリリース：nri.com/jp/news/info/20260224_1.html ／ SCSK ニュース：scsk.jp/news

29. メルカリ Engineering Blog：engineering.mercari.com/blog ／ LINEヤフー Tech Blog：techblog.lycorp.co.jp/ja

30. freee Developers Hub：developers.freee.co.jp ／ CyberAgent Developers Blog：developers.cyberagent.co.jp/blog