Givery 生成AI実務研修 ｜ 協栄産業株式会社 様

Claude Code で進める AI駆動開発

部品在庫ダッシュボードを題材に、仕様駆動開発とテスト駆動開発を2日間で手の内に入れる

本日の進行

講師 安田 光喜（Givery）

AI駆動開発の研修・伴走を担当。本研修の教材とハンズオンを設計

進め方は「手を動かす時間」を中心に置きます。各ハンズオンの前に、そのテーマの要点をこのスライドで短く押さえてから演習に入ります。詳しい背景はオリエンテーション座学編（配布資料）を参照してください。
共有は「講師が指名して画面共有」「チャットに1行」の2つだけ。作業はすべて個人ワークです。詰まったら遠慮なく声をかけてください。

2日間で持ち帰るもの

- Claude Code を VS Code の統合ターミナルで使いこなす
- Skills / Subagents / Commands / Hook の使い分け
- 仕様駆動開発（Spec Kit / Kiro 式）とテスト駆動開発
- 自分の業務に持ち帰れる、動くWebアプリと再利用テンプレ

2日間のアジェンダ

日	午前	午後
Day1	基礎と最新動向／基礎演習／コマンド演習	Skills 3段階／Subagents／Spec Kit で仕様化
Day2	Spec Kit 完走／Kiro 式で別プロジェクト／比較	テスト駆動開発／本題材を SDD×TDD で開発／発展課題

各日 **9:00-17:00**。各ハンズオンの直前に「そのテーマの解説スライド」を挟みます。題材は **部品在庫・納期照会ダッシュボード**、学習用に「在庫アラート通知ツール」を Spec Kit と Kiro 式で作ります。

受講前 → 2日後

いま（受講前）

- 補完型AIは使うが、エージェント型は触り始め
- Skills・Subagent・Hook は名前を知る程度
- AIの出力をどこまで信じてよいか勘所がない

2日後（ゴール）

- 仕様→テスト→実装を Claude Code で回せる
- ハーネス（5部品）で再現性を作れる
- 動くWebアプリを自分の業務に転用できる

DAY 1 | S01 座学

まず現在地と道具の素性を揃える

最新動向 → Claude Code 基礎 → セキュリティの線引き

3系統の役割分担（2026年6月時点）

系統	代表モデル	強み	位置づけ
Claude	Opus 4.7 / Sonnet 4.6 / Haiku 4.5	長尺エージェント・複数ファイル横断・指示への忠実さ	Agentic Coding の本命。本研修の主役
GPT	GPT-5 系 / Codex	汎用対話・マルチモーダル	Codex CLI が直接競合
Gemini	Gemini 3 系	超長文・Workspace連携	社内文書連携で選ばれる

「どれが一番か」ではなく「どの仕事をどれに振るか」。コーディングは長時間任せたときの破綻しにくさで Claude が選ばれています。

出典: **Claude Models Overview** platform.claude.com/docs/en/about-claude/models/overview ／ **OpenAI Codex** openai.com/codex ／ **Gemini** deepmind.google/models/gemini（当日に最新確認）

モデルの外側を組む 5 部品（ハーネス）

Skills

手順とノウハウの再生。`.claude/skills/` 。自動/手動発火

Subagents

別コンテキストの専任。並列可。`.claude/agents/`

Commands

定型プロンプトのワンキー化。`/name`

Hook

イベントで外部コマンド発火。安全装置・自動化

MCP

外部ツール接続の規格。読み取りから始める

CLAUDE.md

プロジェクト規約のシングルソース。起動時に自動で読む

この5部品を Day1 で **1つずつ実物を動かして**覚えます。各部品の演習に入る前に、このあと個別の解説スライドを挟みます。

出典: **Claude Code Docs** code.claude.com/docs

リスクは3層で線を引く

層	何が起きるか	主な対策
入力リスク	顧客データ・ソース・APIキーの平文投入	データ分類・入力前チェック・契約条項の確認
出力リスク	ハルシネーション・脆弱なコードをそのまま採用	レビュー必須・テスト検証・出典の裏取り
運用リスク	与えた権限で本番破壊・無制限書き込み	Permission 設計・Hook で危険コマンド遮断・環境分離

事故の多くは「AIの暴走」ではなく「人が線引きを決めていなかった」こと。納品したコードの責任は納品者にあります。見本テンプレに **Semgrep + gitleaks** の二段防御を同梱しています。

出典: **IPA セキュリティ** ipa.go.jp/security ／ **経産省 AI事業者ガイドライン** meti.go.jp ／ Semgrep+gitleaks は zenn.dev pesca 記事

例：危険コマンドを止める Hook

```
// .claude/settings.json
{ "hooks": {
  "PreToolUse": [{
    "matcher": "Bash",
    "hooks": [{ "type": "command",
      "command": "bash .claude/hooks/block-dangerous.sh" }]
  }]
}}
# block-dangerous.sh: rm -rf / push --force を検出したら exit 2
```

いつ発火するか

`PreToolUse` ＝ツール実行前 / `PostToolUse` ＝実行後 / `Stop` など。前で止めれば実行自体を防げる。

matcher

対象ツール（Bash, Write など）。その操作の前後に外部コマンドを差し込む。

何を書くべきか

人の注意力に頼らない安全装置。危険コマンド遮断・Write後のlint・秘密情報検知。`exit 2` でブロック。

出典: **Hooks** code.claude.com/docs/en/hooks

演習 S02

Claude Code 基礎操作

ショートカット・モデル・設定の階層を、手を動かして体に入れる

会話の状態を操る4つのショートカット

コマンド	役割	使いどころ
`/init`	CLAUDE.md のたたき台を生成	プロジェクト開始時。中身は人が育てる
`/context`	いま何を読んでいるか確認	トークンが膨らんだとき
`/compact`	会話を要約圧縮（流れは残す）	同じ話題を続けるとき
`/clear`	履歴ごと仕切り直し	話題が変わるとき

モデルと Effort

単純作業は速いモデル、設計判断は深いモデル。`/model` で切替。迷ったら標準、重いときだけ上げる。

設定の階層

ユーザー設定 `~/.claude`（全体の土台）をプロジェクト `.claude` が上書き。衝突時はプロジェクトが優先。

注意：コンテキストは「多いほど良い」ではなく「関係が深いほど良い」。無関係な情報は出力品質を下げます（Context Rot）。

演習 S03

コマンド（自作スラッシュコマンド）

既存コマンドを動かし、自分の定型作業を1つワンキー化する

コマンドの正体は「Markdown 1枚」

仕組み

`.claude/commands/<name>.md` を置くだけで `/name` で呼べる。ファイル名がコマンド名、`$ARGUMENTS` で引数を受け取る。

活用例

- `/review-diff` 差分レビュー＋commit案
- `/explain-error` エラー原因の絞り込み
- `/triple-review` 3観点を並列レビュー

ルール・注意点

- 置き場所は `.claude/commands/` 直下（サブフォルダは名前空間が変わる）
- 「補足なしで一発で意図どおり動く」まで本文に前提を書き込む
- 毎回手で足している説明こそコマンド本文へ

出典: **Claude Code Slash Commands** code.claude.com/docs/en/slash-commands

例：`/review-diff` の中身

```
---
description: 変更差分をレビューしcommit案を3つ出す
argument-hint: [対象パス]
---

git の変更差分（$ARGUMENTS）をレビューしてください。
1. リスクの高い変更を上から指摘（影響範囲・想定バグ）
2. テストの要否を判断
3. commit メッセージ案を3つ（簡潔/標準/詳細）
修正はせず、指摘と提案のみ返してください。
```

フロントマター（--- の中）

`description` ＝ `/` の候補一覧に出る説明。`argument-hint` ＝引数の入力ヒント。

本文

普段打っているプロンプトをそのまま書く。`$ARGUMENTS` で渡した引数を受け取る。

何を書くべきか

毎回手で打つ定型指示。「補足なしで一発で意図どおり動く」まで前提を本文に書き切る。

置き場所: **.claude/commands/review-diff.md** ／ ファイル名がコマンド名になる

演習 S04

Skills（3段階）

単機能 → Subagent並列 → 画像アセット読込 と、Skillの幅を観察する

呼ばれたら手順とノウハウを再生する

段階1 単機能

meeting-summary：会議メモを「決定/宿題/未解決」に整形

段階2 並列

multi-review：設計・セキュリティ・バグの3観点を Subagent 並列で集約

段階3 素材付き

ui-from-mock：モック画像を読みHTML骨子を出す（補助ファイル同梱）

ルール

`SKILL.md` の frontmatter に `name` / `description`。description が「いつ使うか」の発火判定になる。自動発火と手動発火 `/skill-name`。

注意点

YAML 構文（コロン後の半角スペース）と ファイル名 `SKILL.md`。description が曖昧だと自動発火しない。個人の暗黙知をチーム資産に変える部品。

出典: **Agent Skills** code.claude.com/docs/en/skills

例：`SKILL.md` の中身

```
---
name: meeting-summary
description: >
  会議メモを「決定/宿題/未解決」に整形する。
  「会議メモを整形」「議事メモまとめて」で発火。
---

# Meeting Summary
受け取ったメモを次の3区分のMarkdownに整形する。
## 決定事項　## 宿題(ToDo)　## 未解決
原文にない内容は作らない。判断不能は未解決へ。
```

フロントマター

`name` ＝スキル名（フォルダ名と一致）。`description` ＝**発火判定**。「いつ使うか」を具体語で書くほど自動発火しやすい。

本文

手順・判断基準・出力フォーマット。補助ファイル（テンプレ等）を同フォルダに置いて参照させてもよい。

何を書くべきか

繰り返す作業の手順。個人の暗黙知をチーム資産に変える単位で切り出す。

置き場所: **.claude/skills/meeting-summary/SKILL.md**（大文字）

演習 S05

Subagents

単体で呼び、次にコマンド起点で3体を並列起動する

親と別コンテキストで動く専任エージェント

使い所

- 観点が独立した仕事の並列化（設計・セキュリティ・パフォーマンス）
- レビューを切り出し、実装の文脈に引きずられない指摘を得る
- コマンド起点で3体同時：`/triple-review`

ルール・注意点

- 定義は `.claude/agents/<name>.md`。責務（何を見て何を見ないか）を書く
- 渡す情報を絞るほど文脈汚染を防げる
- 前の結果を次が使う仕事は並列にしても待ちが出る

出典: **Sub-agents** code.claude.com/docs/en/sub-agents

例：`security-auditor.md` の中身

```
---
name: security-auditor
description: セキュリティ観点のレビュー専任
tools: Read, Grep, Glob, Bash
---

秘密情報の混入・危険コマンド・脆弱な実装だけを見ます。
重要度(High/Medium/Low)順に、根拠と修正方針を
セットで返してください。修正はしません。
```

フロントマター

`name` / `description` に加え、`tools` で使える道具を絞る（与えすぎない）。

本文

責務＝「何を見て、何を見ないか」と、出力の形（並び順・粒度）。

何を書くべきか

役割は1つに絞る。渡す情報を絞るほど文脈汚染を防ぎ、並列で効く。

置き場所: **.claude/agents/security-auditor.md**／コマンドから並列起動（/triple-review）

演習 S06 ｜ 前半

仕様駆動開発（SDD）へ

コードを書く前に「何を作るか」を文書で固める

コードの前に手戻りを潰す 4 段

```
Spec  ──→  Plan  ──→  Tasks  ──→  Implement
何を作るか      どう作るか      実行単位に割る      テスト先行で実装
```

1行の指示で1,000行が出る時代、間違った1,000行のコストは無視できません。受け入れ基準を先に言語化し、そこから機械的に展開します。粒度は Epic ＞ Feature ＞ Story ＞ 受け入れ基準。

出典: **GitHub Spec Kit** github.com/github/spec-kit ／ **Anthropic: Effective context engineering** anthropic.com/engineering

演習 S06 | 後半

Spec Kit ハンズオン

在庫アラート通知ツールを /specify → /plan → /tasks で仕様化

柔軟にカスタムする流派（GitHub）

構成・起動

`.specify/` のテンプレ＋ `/specify` `/plan` `/tasks` `/implement` 。起動は `uvx specify init` 。**本研修は初期化済みを同梱**（社内ネットで取得不可でも動く）。

使い所

固有の規約・複雑な要件を持つ案件。テンプレもコマンドも書き換えられ、constitution で独自ルールを注入できる。

強み	弱み
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	テンプレ整備の初期コスト・運用ルールを自分で設計する必要

出典: **Spec Kit** github.com/github/spec-kit ／ Releases で最新のテンプレ・コマンド変更を追う

コマンドと成果物の流れ

```
uvx specify init .          # 初期化（本研修は同梱済み）

/specify docs/meeting-notes.md   → spec.md
/plan                            → plan.md
/tasks                           → tasks.md
/implement                       → テスト先行で実装→Green
```

各段でやること

/specify 議事メモから受け入れ基準つき仕様。**/plan** 技術選定・構成（理由つき）。**/tasks** テスト作成を先頭に15分粒度。**/implement** テスト先行で実装。

確認すべきこと

未確定（送信先・頻度など）を勝手に埋めていないか。tasks の先頭がテスト作成になっているか。

テンプレは **.specify/** で書き換え可（柔軟カスタムの思想）／ github.com/github/spec-kit

DAY 2

完走 → 比較 → テスト駆動 → 本題材

Spec Kit を仕上げ、同じ要件を Kiro 式で作り、TDD で固める

演習 S01

Spec Kit 完走

テスト先行 → /implement で Green → 追加機能を仕様から積み増す

演習 S02

Kiro 式ハンズオン

同じ要件を固定3ファイルで、完全な別プロジェクトとして作る

固定3ファイルで迷いを減らす流派（AWS）

構成・起動

`requirements.md` / `design.md` / `tasks.md` の3ファイル固定。Kiro IDE の思想を、本研修では **kiro-spec Skill** で Claude Code 上に再現（別プロジェクトで実施）。

使い所

定型的な機能開発、立ち上げ初期のチーム。「誰がやっても同じ手順」で属人化しにくい。

強み	弱み
学習コストが低い・型が決まる・属人化しにくい	固有事情への柔軟性・複雑系の表現力に限界

出典: **Kiro Specs** kiro.dev/docs/specs ／ Changelog で更新を追う

例：`requirements.md`（EARS形式）

```
# requirements
- 在庫が発注点を下回った場合、システムは
  その部品を検知すること。
- 検知した場合、システムは 型番・現在庫・
  発注点・不足数 を含む通知文を生成すること。
- 在庫が十分な場合、システムは通知しないこと。
```

3ファイルの役割

requirements.md 受け入れ基準（EARS=「〜の場合、システムは〜すること」）。**design.md** モジュール構成・機能と関数の対応。**tasks.md** テスト作成を先に、依存つき。

何を書くべきか

受け入れ基準を漏れなく。曖昧な「Then」は具体値に直す。型が決まっているので迷いが少ない。

Spec Kit と Kiro 式 — 同じ要件で作って比べる

観点	Spec Kit	Kiro 式
思想	柔軟にカスタム	固定3ファイルで迷いを減らす
学習コスト	中	低
向く案件	固有規約・複雑要件	定型開発・立ち上げ初期
属人化	テンプレ設計者に依存しやすい	低い

同じ「在庫アラート通知ツール」を別プロジェクトで2回作るので、違いが机上比較ではなく手の実感として残ります。最後に「自社の案件タイプならどちらか」を1つ決めます。

演習 S04 → S05

テスト駆動開発（TDD）

受け入れ基準をテストにし、実装の正しさを機械で固定する

Red → Green → Refactor

01

Red

失敗するテストを先に書く。これが「完成の定義」

02

Green

通す最小の実装。まず動かす

03

Refactor

緑を保ったまま整える。壊したらすぐ赤で気づく

> テスト＝受け入れ基準を機械が判定できる形にしたもの
>
> だから Claude に実装を任せても Green かどうかで完成を客観判定できる。SDD の受け入れ基準と地続き。配布フォルダの tdd_setup でテスト環境を確認してから本題材へ。

演習 S06

本題材開発

動くWebアプリに、SDD×TDD で機能を1つ足し切る

部品在庫・納期照会ダッシュボード

架空企業・ダミーデータ。営業寄りの方は別トラックの提案アシスタントを選べます。

電子部品商社の現場を模した、ブラウザで動くWebアプリ。検索・在庫薄ハイライト・納期ソートが動きます。

- ロジックは `src/inventory.mjs` に分離（テスト可能）
- 足す機能例：発注リストCSV出力・グラフ・取込
- docs/ の demo-mtg で進め方を先に掴む

技術営業 提案アシスタント

ヒアリングメモを貼ると、課題整理・追加質問・提案方針・構成案・メール・提案書を組み立てます。

- 生成ロジックは決め打ちで動く＝テストで固定できる
- 拡張：業種別テンプレ・過去案件参照・docx出力

持ち帰って効く仕掛け

すぐ試すテンプレ + 見本ZIP

`.claude/` に即試すテンプレ。実務用の完成形は見本ZIPで後から配置。Kiro再現Skill・Semgrep+gitleaks Hook も同梱

会議を覗くデモ（demo-mtg）

各題材の docs/ に「どう進めるか」を会話形式で。初見でも輪郭を掴める

チーム内可視化プロンプト

docs/README のプロンプトを実行すると、題材を5分で理解できるHTMLが生成される

出典と安全の明示

各テンプレ末尾に「ハーネス種別 / 講師 安田によるセキュリティチェック済 / 引用元URL」

協栄産業として次に踏む3歩

1

共通CLAUDE.md / Skill

社内で1つ作り、書き方をそろえる

2

Spec Kit / Kiro を試す

1ヶ月運用して比較データを取る

3

Hook で安全を機械化

危険コマンド遮断・秘密情報検知を仕込む

Spec を書く・ハーネスを組む・TDD で固定する・最新を自分の仕組みで追う

この4つを現場の習慣にできれば、研修の元は取れます。

参考にした一次情報

Claude Code Docs
code.claude.com/docs

Claude Models / Pricing
platform.claude.com/docs/en/about-claude/models/overview

Anthropic Engineering Blog
anthropic.com/engineering

GitHub Spec Kit
github.com/github/spec-kit

Kiro Specs（AWS）
kiro.dev/docs/specs

Model Context Protocol
modelcontextprotocol.io

IPA セキュリティ
ipa.go.jp/security

経産省 AI事業者ガイドライン
meti.go.jp

Semgrep + gitleaks 自動スキャン（pesca）
zenn.dev/zittiandbuoni/articles/632ff0709247f6

Anthropic knowledge-work-plugins
github.com/anthropics/knowledge-work-plugins

モデル名・対応表は更新が速い領域です。当日に最新を確認してから提示します。