Givery, Inc.

HANDS-ON GUIDE DAY 2

ハンズオンガイド Day 2

生成AI実務研修（2 日間）
受講者用 Day 2 進行ガイド

提供　Givery 株式会社

配布先　協栄産業株式会社 御中

形式　VS Code × Claude Code（CLI）、個人ワーク

版数　v2.0（2026 年 6 月 20 日）

本書は Day 2 の進行ガイドです。Day 1 で着手した Spec Kit を完走し、同じ要件を Kiro 式で別プロジェクトとして作って比較、その後テスト駆動開発を学び、メイン題材「部品在庫・納期照会ダッシュボード」（営業寄りの方は別トラック「技術営業 提案アシスタント」）を仕様駆動 × テスト駆動で開発します。ヒントは `hints/`、参考解は `_reference/` にあります。

1. 本日のゴール

Day 2 が終わるとき、受講者は次の状態に到達しています。

- Day 1 で 3 段（spec / plan / tasks）まで進めた在庫アラート通知ツールを、Spec Kit で `/implement` まで完走し、追加機能 3 ステップまで進めている
- 同じ要件を Kiro 式（`requirements.md` / `design.md` / `tasks.md` の固定 3 ファイル）で、完全な別プロジェクトとして 4 ステップ実装している
- Spec Kit と Kiro 式の違いを、自分が両方作った実体験として観点別に説明できる
- テスト駆動開発（Red → Green → Refactor）の流れと強みを理解し、配布フォルダの TDD 設定を使って 1 周自分の手で回している
- メイン題材「部品在庫・納期照会ダッシュボード」（または別トラックの提案アシスタント）に、仕様駆動 × テスト駆動で機能を足し、追加機能のテストが Green かつブラウザで動作している
- README を整備し、最終 commit と tag を打って持ち帰れる状態になっている

---

2. タイムテーブル [420min]

時間は分数のみで示します（昼休憩 [60min] を除く実働 [420min]）。

枠	時間	内容
S01	[90min]	Day 1 の復習 + Spec Kit 完走（`/implement` + 追加機能 3 ステップ）
S02	[80min]	Kiro 式ハンズオン（同じ要件を別プロジェクトで 4 ステップ）
S03	[30min]	Spec Kit と Kiro 式の比較
昼休憩	[60min]	–
S04	[40min]	テスト駆動開発の基礎（座学 + 配布フォルダの TDD 設定確認）
S05	[50min]	テスト駆動開発の実践（純粋関数で Red → Green → Refactor を 1 周）
S06	[90min]	メイン題材開発（部品在庫ダッシュボードに SDD×TDD で機能追加・ブラウザ確認）
S07	[40min]	発展課題 + 質疑応答・振り返り・閉会

各 Phase の終わりに「ここまでで詰まった点」を研修チャットに 30 秒で投下してください。講師がリアルタイムで拾います。演習はすべて個人ワークです。

---

3. ヒントと参考解の使い方

各演習には通番付きのヒントファイルがあります。3 分考えて手が止まったら、該当するヒントを開いてください。

対象演習	ヒントファイル
Spec Kit 完走（S01）	`hints/s06_speckit_hint.md`（Day 1 と共通・続き）
Kiro 式ハンズオン（S02）	`hints/s02_kiro_hint.md`
TDD 実践（S05）	`hints/s05_tdd_hint.md`
メイン題材開発（S06）	`hints/ho09_capstone_hint.md`

> **注意**
>
> 配布 ZIP の `_reference/` フォルダには講師が作成した参考解（在庫アラートの完成形と、技術営業支援エージェントの完成版）が入っています。研修中は開かないでください。自分で Red から Green まで到達する体験が Day 2 の中心です。持ち帰り後の答え合わせに使ってください。

---

4. S01：Day 1 の復習 + Spec Kit 完走 [90min]

4.1 復習 [20min]

Day 1 で触れた道具を、メインテーマで使う前に手元で温め直します。次を各 1 回ずつ実行してください。

- `speckit-inventory/` で Claude Code を起動し、`/context` で Day 1 の到達点（spec.md / plan.md / tasks.md）を確認する
- Day 1 で作った自作コマンド（`day1/s03_commands/.claude/commands/`）を 1 回呼ぶ
- `/triple-review` を 1 回呼び、3 体並列の動きを思い出す

復習の最後に、Day 1 終了時の宿題（自分の業務で AI に任せたい定型作業 1 つ）をチャットに 1 行投下してください。S06 と S07 で使います。

4.2 Spec Kit 完走 [70min]

HANDS-ON 01　在庫アラート通知ツールを Spec Kit で完成させる

学習目標	Day 1 で固めた spec / plan / tasks から `/implement` で実装を完走し、追加機能を 3 ステップ進める
作業フォルダ	`kyos-handson/speckit-inventory/`（Day 1 の続き）
到達点	テストが Green の状態で 3 機能（監視・閾値判定・通知生成）が動き、追加機能が 1 つ入っている
ヒント	`hints/s06_speckit_hint.md`

STEP 1 ── 実行 [20min]

テストを先に置いてから `/implement` に入る

tasks.md の先頭グループ（テスト作成）から進めます。まずテストだけを書かせ、Red を確認してから実装に入ります。AI への指示例：

```
tasks.md のテスト作成タスクに従って、受け入れ基準を検証する pytest のテストを tests/ に書いてください。
テストだけ書いて、実装はまだしないでください。src/ 配下の作成・編集は禁止です。
書き終えたら python -m pytest tests/ -v を実行して、全件失敗（Red）であることを確認してください。
```

確認方法：自分でも `python -m pytest tests/ -v` を実行し、Red を目視します。実装がなくても通るテストがあれば、それは検証として機能していないので書き直しを依頼します。

STEP 2 ── 実行 [25min]

`/implement` で Green まで持っていく

`/implement` を実行し、tasks の実装タスクを順に進めさせます。スコープを縛るのが要点です。

```
/implement tasks.md の実装タスクを順番に進めてください。
tests/ 配下の変更は禁止です。1 タスク終わるごとに python -m pytest tests/ -v を実行し、
通ったテストの数を報告してください。テストが全件 Green になったら止めてください。
```

確認方法：自分でも pytest を実行して Green を確認したうえで、実装を通読します。テストを通すためだけの不自然な分岐（期待値のハードコード等）がないかを見てください。

STEP 3 ── 実行 [25min]

追加機能を 3 ステップ進める

完成した spec に小さな機能追加を 1 つ載せ、「spec 追記 → `/plan` 差分 → `/tasks` 差分」の流れで仕様から積み増します。例：通知メッセージに在庫数を括弧書きで添える、閾値を商品カテゴリ別に持てるようにする。

```
spec.md に「通知メッセージに現在の在庫数を括弧書きで添える」要件を追記してください。
その後 /plan と /tasks の差分だけを更新し、追加されたタスクを一覧で示してください。
追加分のテストを先に書いて Red を確認してから実装してください。
```

到達点の確認：追加機能のテストも含めて全件 Green になれば S01 完了です。ここまでで「コードを直接いじらず、仕様から積み増す」感覚が身についていれば狙いどおりです。

早く終わったら：もう 1 つ機能を追加するか、code-reviewer Subagent でレビューを 1 回かけて指摘を 1 件反映してください。

---

5. S02：Kiro 式ハンズオン [80min]

同じ「在庫アラート通知ツール」を、今度は Kiro 式で**完全な別プロジェクト**として作ります。要件は同じ、進め方だけが違う。これが S03 の比較の土台になります。Kiro 式は `requirements.md` / `design.md` / `tasks.md` の固定 3 ファイルで進めます（Kiro IDE の思想を Claude Code 上で再現します）。作業フォルダは `kyos-handson/kiro-inventory/` です。

HANDS-ON 02　同じ要件を Kiro 式の 4 ステップで実装する

学習目標	固定 3 ファイルのパイプラインで、要求 → 設計 → タスク → 実装を 4 ステップで通す
作業フォルダ	`kyos-handson/kiro-inventory/`（Spec Kit と同じ `docs/meeting-notes.md` 同梱）
到達点	requirements.md / design.md / tasks.md が揃い、テスト Green で 3 機能が動く別プロジェクト
ヒント	`hints/s02_kiro_hint.md`

STEP 1 ── 実行 [20min]

requirements.md（要求）を EARS 形式で書く

同じ議事メモから、まず requirements.md だけを作ります。design / tasks はまだ作りません。AI への指示例：

```
docs/meeting-notes.md をもとに、Kiro 式の requirements.md を
プロジェクト直下に作成してください。EARS 形式（〜の場合、システムは〜すること）で書いてください。
スコープは Spec Kit 版と同じ 3 機能（監視・閾値判定・通知生成）です。
design.md と tasks.md はまだ作らないでください。
```

確認方法：受け入れ基準が「〜の場合、システムは〜すること」の形で並んでいるか、Spec Kit 版の spec と同じ機能範囲かを確認します。

STEP 2 ── 実行 [20min]

design.md（設計）を起こす

```
requirements.md を入力に、Kiro 式の design.md を作成してください。
モジュール構成、データの流れ、3 機能と関数の対応表を含めてください。
コードはまだ書かないでください。
```

確認方法：requirements の各項目が design のどのモジュールに落ちるか、自分でトレースできることを確認します。

STEP 3 ── 実行 [15min]

tasks.md（タスク）に割る

```
design.md をもとに tasks.md を作成してください。
各タスクに依存タスクを明記し、テスト作成タスクを実装タスクより先に並べてください。
末尾に依存関係をテキストの矢印図で示してください。
```

確認方法：テスト作成が先頭に来ているか、1 タスクが 15 分以内のサイズかを確認します。

STEP 4 ── 実行 [25min]

実装して Green にする

tasks の順にテスト先行で実装します。Spec Kit の `/implement` に当たる工程を、通常のプロンプトで進めます。

```
tasks.md のテスト作成タスクから進めてください。まずテストだけ書いて Red を確認し、
その後に実装して Green にしてください。tests/ の変更は実装フェーズでは禁止です。
全件 Green になったら python -m pytest tests/ -v の結果を報告してください。
```

到達点の確認：Kiro 版のプロジェクトでテストが全件 Green になれば S02 完了です。Spec Kit 版（`speckit-inventory/`）とは別フォルダに、同じ機能のもう 1 つの実装が並んでいる状態になります。

早く終わったら：Spec Kit 版で入れた追加機能（在庫数の括弧書きなど）を Kiro 版にも入れ、同じ変更が両方式でどう違って進むかを見比べてください。

---

6. S03：Spec Kit と Kiro 式の比較 [30min]

2 つの完成プロジェクトを並べ、観点ごとに違いを言語化します。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を判断できる状態がゴールです。

手順 [20min]

`speckit-inventory/` と `kiro-inventory/` を並べて開き、次の 5 観点で 1 行ずつメモを取ってください。同じ要件を 2 通りで作った自分の体験が、そのまま比較データになります。

観点	見るところ
立ち上がりの速さ	最初の仕様ができるまでの手数。コマンド主導（Spec Kit）と素の対話（Kiro 式）でどう違ったか
迷いの少なさ	「次に何をすればいいか」で止まった回数。固定 3 ファイルが効いたか、自由度が負担になったか
カスタムのしやすさ	プロジェクト固有の事情を書き込む場所があったか。テンプレを直せたか
成果物の構造	生成された文書とコードの整い方。後から読み返しやすいのはどちらか
属人化リスク	「自分以外の人が同じ手順で再現できそうか」

まとめ [10min]

講師が「どの場面で Spec Kit、どの場面で Kiro 式」という使い分けマトリクスを提示します。自分のメモと突き合わせ、社内に持ち帰る判断基準を 1 つ決めてチャットに投下してください。

> **Tips**
>
> 一般論としては、固有の規約や複雑な要件を抱える案件は Spec Kit、定型的な機能開発や立ち上げ初期のチームは Kiro 式が向きます。ただし最後は自分が両方触った実感で決めてください。

---

7. S04：テスト駆動開発の基礎 [40min]

7.1 テスト駆動開発とは [25min]

テスト駆動開発（TDD）は、実装より先にテストを書く開発スタイルです。流れは 3 拍子です。

01 **Red —— 失敗するテストを先に書く**

これから作る機能の「正しさ」をテストの形で先に定義します。実装がまだないので、テストは必ず失敗します。この赤が「何を満たせば完成か」の宣言になります。

02 **Green —— テストが通る最小の実装をする**

テストを通すことだけを目標に、最短で実装します。きれいさは後回しでよい。まず動く状態（緑）を作ります。

03 **Refactor —— 緑を保ったまま整える**

テストが通る状態を維持したまま、コードを読みやすく直します。テストがあるので、壊したらすぐ赤で気づけます。

TDD の強みは AI 駆動開発と相性が良い点にあります。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（spec の受け入れ基準）とテスト駆動開発（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。

7.2 配布フォルダの TDD 設定を確認する [15min]

S05・S06 で使う TDD の環境を、先に確認しておきます。`kyos-handson/day2/tdd_setup/` を開き、次を確認してください。

- `pytest.ini`（または `pyproject.toml` の `[tool.pytest.ini_options]`）でテスト探索パスが `tests/` に設定されている
- `tests/` と `src/` がフォルダで分かれている
- サンプルテスト 1 件があり、`python -m pytest tests/ -v` が通る
- `.claude/settings.json` に「Write 直後にテストを促す」運用メモが入っている（Hook の雛形は hints 参照）

確認方法：`cd kyos-handson/day2/tdd_setup && python -m pytest tests/ -v` でサンプルテストが PASSED になれば、環境は準備完了です。

---

8. S05：テスト駆動開発の実践 [50min]

HANDS-ON 03　純粋関数で Red → Green → Refactor を 1 周回す

学習目標	受け入れ基準 → テスト列挙 → Red → 実装 → Green → Refactor の 1 周を、Claude Code と一緒に回す
作業フォルダ	`kyos-handson/day2/tryit_tdd/`（サンプル日報 `data/daily_report_sample.txt` 同梱）
作るファイル	`spec/parse_daily_report.md` ／ `tests/test_parse_daily_report.py` ／ `src/parse_daily_report.py`
ヒント	`hints/s05_tdd_hint.md`

STEP 1 ── 考える [8min]

受け入れ基準を spec に書く

`data/daily_report_sample.txt` を開いて眺めてください。「案件名 工数(h)」形式の行が混ざった自由記述の日報です。何を入力に取り、何を返す関数なら「正しい」と言えるかを決め、`spec/parse_daily_report.md` に受け入れ基準を 3〜5 件書きます。

> **Tips**
>
> 受け入れ基準は「〜の場合、〜を返す」の形で書くと、そのままテストケースになります。空文字・工数 0・同一案件の複数行など、境界になりそうな入力を 1 つは入れてください。

STEP 2 ── 書く [12min]

テストを列挙し、Red を確認する（実装はまだ書かない）

```
spec/parse_daily_report.md を読んでください。
受け入れ基準を満たすかを検証する pytest のテストを tests/test_parse_daily_report.py に書いてください。
テスト対象は src/parse_daily_report.py の関数 parse_daily_report(text: str) -> dict[str, float] です。
テストだけ書いて、実装はまだしないでください。src/ 配下のファイルは作成も編集も禁止です。
```

ターミナルで `python -m pytest tests/ -v` を実行し、全テストが FAILED または ERROR になることを確認します。この「全部赤」が Red です。

STEP 3 ── 実行 [18min]

実装して Green にし、リファクタする

```
tests/test_parse_daily_report.py がすべて通るように
src/parse_daily_report.py に parse_daily_report を実装してください。
テストファイルの変更は禁止です。実装後に python -m pytest tests/ -v を実行して
全件 PASSED になることを確認してください。
```

Green を確認したら、続けて「テストを通したまま、関数を読みやすくリファクタしてください」と依頼し、リファクタ後も Green が維持されることを確認します。自分でも pytest を実行し、Claude の報告と一致するか目視で照合してください。

STEP 4 ── 答え合わせ [12min]

もう 1 周回す

spec に受け入れ基準を 1 件追加し（例：工数の合計が 24h を超えたら ValueError）、テスト追加 → Red → 実装 → Green の 1 周をもう一度回します。2 周目で TDD のリズム（赤を出してから緑にする往復）が手に馴染みます。

早く終わったら：Subagent で「実装役」と「テストレビュー役」を分け、テストの抜けを別観点で点検させてください。

9. S06：メイン題材開発 [90min]

本研修の山場です。**メイン題材は「部品在庫・納期照会ダッシュボード」**（`apps/parts-dashboard/`）。ブラウザで動くWebアプリに、仕様駆動開発（Day1・Day2で身につけた進め方）とテスト駆動開発（S05の往復）で**機能を1つ足し切り、画面で動作確認まで**持っていきます。営業寄りの方は**別トラック「技術営業 提案アシスタント」**（`apps/proposal-assistant/`）を選んで構いません。方式は Spec Kit / Kiro 式のどちらでも構いません。

9.1 題材の掴み方（docs/ を先に見る）

着手前に、選んだ題材の `docs/` を開いてください。背景・データ・進め方が入っています。

- `docs/demo-mtg/index.html` … この題材をどんな流れで進めるか「会議を覗く」デモ（ブラウザで開く）
- `docs/meeting-notes.md` / `docs/company-profile.md` … 発端の打ち合わせと会社情報（架空・ダミー）
- メインA: `docs/data/parts.csv`（部品マスタ）／ 別トラックB: `../data/hearing-sample.mjs`（ヒアリングメモ）

すべて架空・ダミーデータです。実在の企業・在庫・案件ではありません。

9.2 開発手順（メインA：機能を1つ SDD×TDD で足す）

HANDS-ON 04　ブラウザ題材に機能を1つ足し切る

作業フォルダ	メイン `apps/parts-dashboard/`（別トラック `apps/proposal-assistant/`）
足す機能の例	在庫薄の部品を「発注リスト」としてCSV出力する（他に納期別の件数グラフ、CSV取込 など）
最低ライン	追加機能のテストが Green、かつブラウザ画面で新機能が動く
ヒント	`hints/ho09_capstone_hint.md` ／ 取り組み方は `exercise/day2_ho04_capstone.md`

PHASE 1 —— 仕様 [15min]

足す機能を1つ決め、受け入れ基準を書く

docs/ で背景を掴み、足す機能を1つ選びます。選んだ方式（Spec Kit / Kiro 式）で spec（または requirements）に受け入れ基準を Given-When-Then で書きます。例「在庫薄の部品だけを 型番,在庫,発注点,不足数 のCSV文字列にして返す」。

PHASE 2 —— テスト先行（Red）[20min]

src/ の純粋関数に対するテストを先に書く

ロジックは `src/inventory.mjs`（メインA）/ `src/proposal.mjs`（B）に足します。実装の前にテストを書き、Red を確認します。

```
src/inventory.mjs に追加する関数 toReorderCsv(parts) の受け入れ基準を満たす
テストを tests/ に書いてください。テストだけ。実装はまだしないでください。
書いたら node --test tests/ で失敗（Red）を確認してください。
```

PHASE 3 —— 実装（Green）[25min]

純粋関数を実装してテストを通す

テストが通る最小実装を `src/` に足します。`node --test tests/インデックス.test.mjs` が全件 Green になるまで。tests は触りません。

PHASE 4 ── 画面に繋いで動作確認 [25min]

app.js から呼び出し、ブラウザで動かす

追加した関数を `app.js` から呼び、ボタンや表示を足します。そのフォルダで `python3 -m http.server` を起動し、ブラウザで新機能を操作して確認します（`file://` 直開きはモジュール制限で動かない場合があります）。

到達点の確認：追加機能のテストが Green、かつブラウザ画面で実際に動く。ここまで来たら S07 の発展課題へ。

---

10. S07：発展課題 + 振り返り・閉会 [40min]

10.1 発展課題（早く終わった人）[20min]

メインテーマが最低ラインに届いた人は、次から 1 つ選んで進めてください。自由提案も歓迎します。Subagent を使った並列拡張も推奨します。

- 残り 2 機能を Green まで完成させる
- 「提案メールテンプレ生成」を S04 で学んだ Skill として切り出し、エージェントから呼び出す
- Day 1 終了時に持参した「自分の業務で AI に任せたい定型作業」をエージェントに適用する。基盤はそのまま、CLAUDE.md と Skill の差し替えだけで自分の業務向けに転用できるかを試す
- CRM の CSV を読み込んで顧客プロファイルを補完する機能を追加する
- 簡易提案書を docx 形式でも出力する

最低ラインに届いていない人は、発展課題には進まず、`hints/s06_capstone_hint.md` を使って 4 機能 Green の完成を優先してください。詰まったら手を挙げてください。講師が巡回して支援します。

10.2 成果物の整理 [5min]

配布の `README.md` 雛形を埋め、最終 commit と tag を打ちます。

```
git add -A
git commit -m "Day2 完了: 部品在庫ダッシュボード 機能追加"
git tag day2-final
```

10.3 振り返り・質疑応答・閉会 [15min]

2 日間で持ち帰るものを 1 枚に落とします。チャットへ次の 3 行を投下してください。

- 明日から最初に試すこと 1 つ
- Spec Kit / Kiro 式のどちらを自分の業務で使いたいか、理由 1 行
- もっと深掘りしたいテーマ 1 つ

最後に教材サイトの閲覧案内とアンケートに回答して閉会です。

---

11. 提出物（Day 2 終了時）

提出方法は当日アナウンスします。

- 在庫アラート通知ツールの Spec Kit 版（`speckit-inventory/`、テスト Green + 追加機能）
- 在庫アラート通知ツールの Kiro 式版（`kiro-inventory/`、テスト Green）
- Spec Kit / Kiro 式の比較メモ（5 観点 × 1 行）
- メイン題材アプリ（`apps/parts-dashboard/` または `apps/proposal-assistant/`）に追加機能、テスト Green、ブラウザで動作、tag `day2-final`
- README（何を作ったか、動かし方、拡張した点）

---

12. よくある質問（Day 2 特有のもの）

Q1：`/implement` が途中で止まる／空回りする

失敗中のテストが多すぎる可能性があります。`/clear` で文脈を仕切り直し、失敗している 1 件だけに絞って依頼してください。Red の情報量（期待値の具体性）が高いほど、修正の手数が減ります。

Q2：Kiro 式と Spec Kit で成果物が違ってよいのか

問題ありません。同じ要件でも進め方が違えば構造は変わります。受け入れ基準を満たし、テストが Green なら両方とも正解です。違いそのものが S03 の比較材料です。

Q3：テストが Red のまま実装が進まない

受け入れ基準の入出力が曖昧なケースが大半です。spec（または requirements）に戻り、Given-When-Then の「Then」を具体的な値で書き直してください。仕様 → テスト → 実装の順に直すのが SDD × TDD の作法です。

Q4：時間内に 6 機能終わらない

4 機能 Green が最低ラインです。残りは `_reference/` を答え合わせに使いながら持ち帰り後に完成させてください。

Q5：Spec Kit の `specify init` が失敗する

uvx が外部レジストリへ通信できない環境では失敗します。配布済みの初期化フォルダ（`speckit-inventory/`）をそのまま使ってください。手順は `hints/s06_speckit_hint.md` にあります。

---

13. 詰まったときの順番

手が止まったら次の順で解決を試みてください。

- 該当する `hints/` の通番ヒントを開く（3 分考えてから）
- Claude Code 自身にエラーメッセージを貼って原因を聞く
- 挙手して講師を呼ぶ

`_reference/` は研修中は開かない約束です。持ち帰り後の答え合わせまで取っておいてください。

文書末尾。