生成AI利活用ワークショップ ～ 未来の技術者必修! 生成AI活用とAI駆動開発の基礎を1.5日で完全習得 ～

Part 5

開発実践 — 仕様書から、動くアプリまで

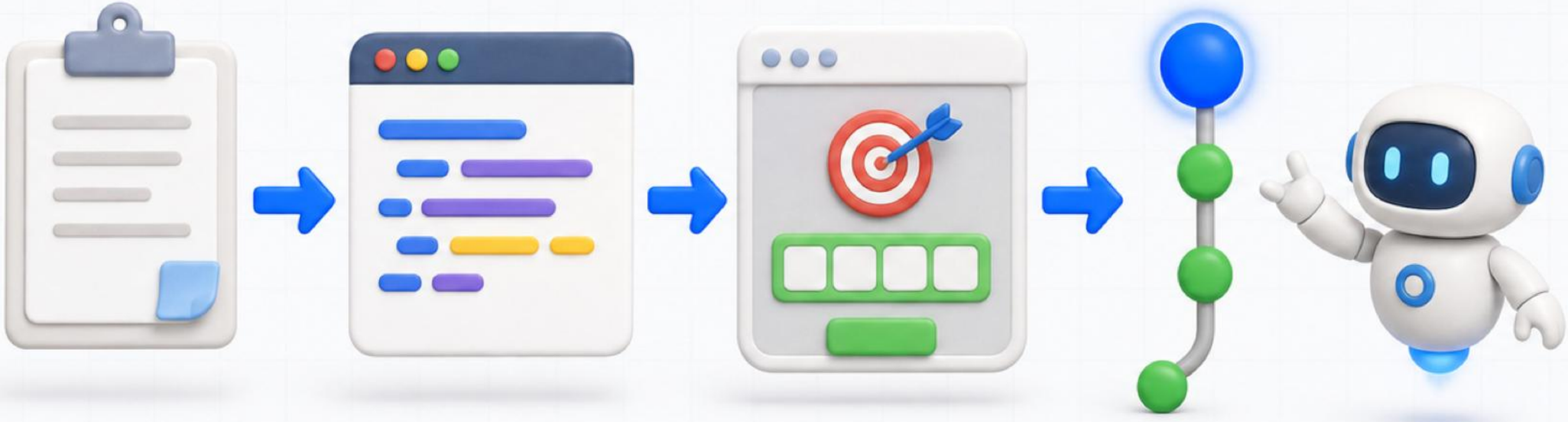
1日目 16:30～18:00 ／ 2日目 9:00～10:45



今日: AI駆動開発の型を、
全員で体得する

明日: その型で、
自分のアプリを仕上げて push する

必ず持ち帰るもの:
仕様書 → 生成 → 検証 → コミット



AI駆動開発のワークフロー全体像

1 仕様書を書く —
何を作るかを、
自分の言葉で決める



2 生成させる —
エージェントに
コードを書かせる



3 検証する —
実行して、差分を読んで、
確かめる



4 コミットする —
動いた状態を
セーブする

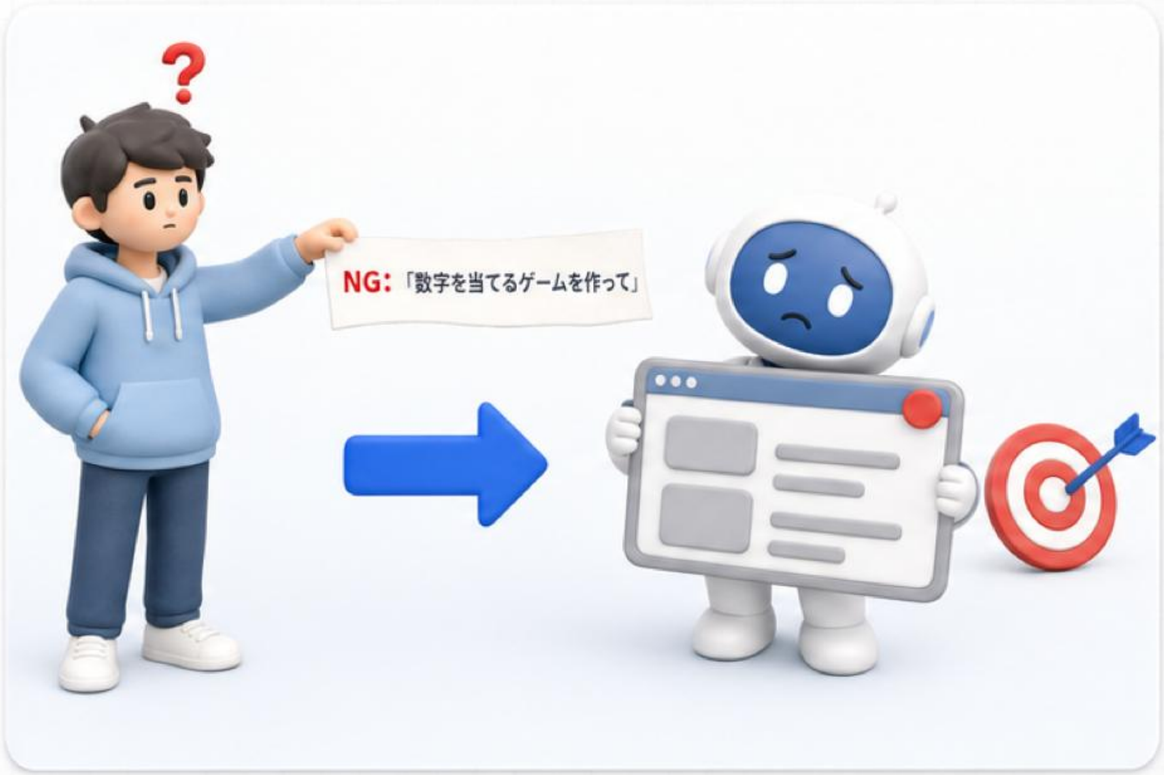


→ 1 に戻る
（機能追加は、
ループをもう1周）

**このループを
何周回せるかが、
開発の速さ**

なぜ仕様書を先に書くのか

曖昧な依頼 → 曖昧なアプリ





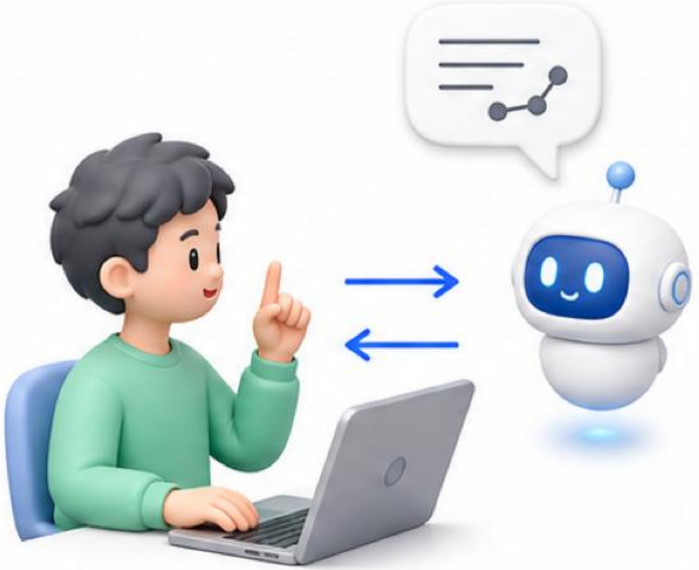
要点:

仕様書は、AIへのコンテキスト設計

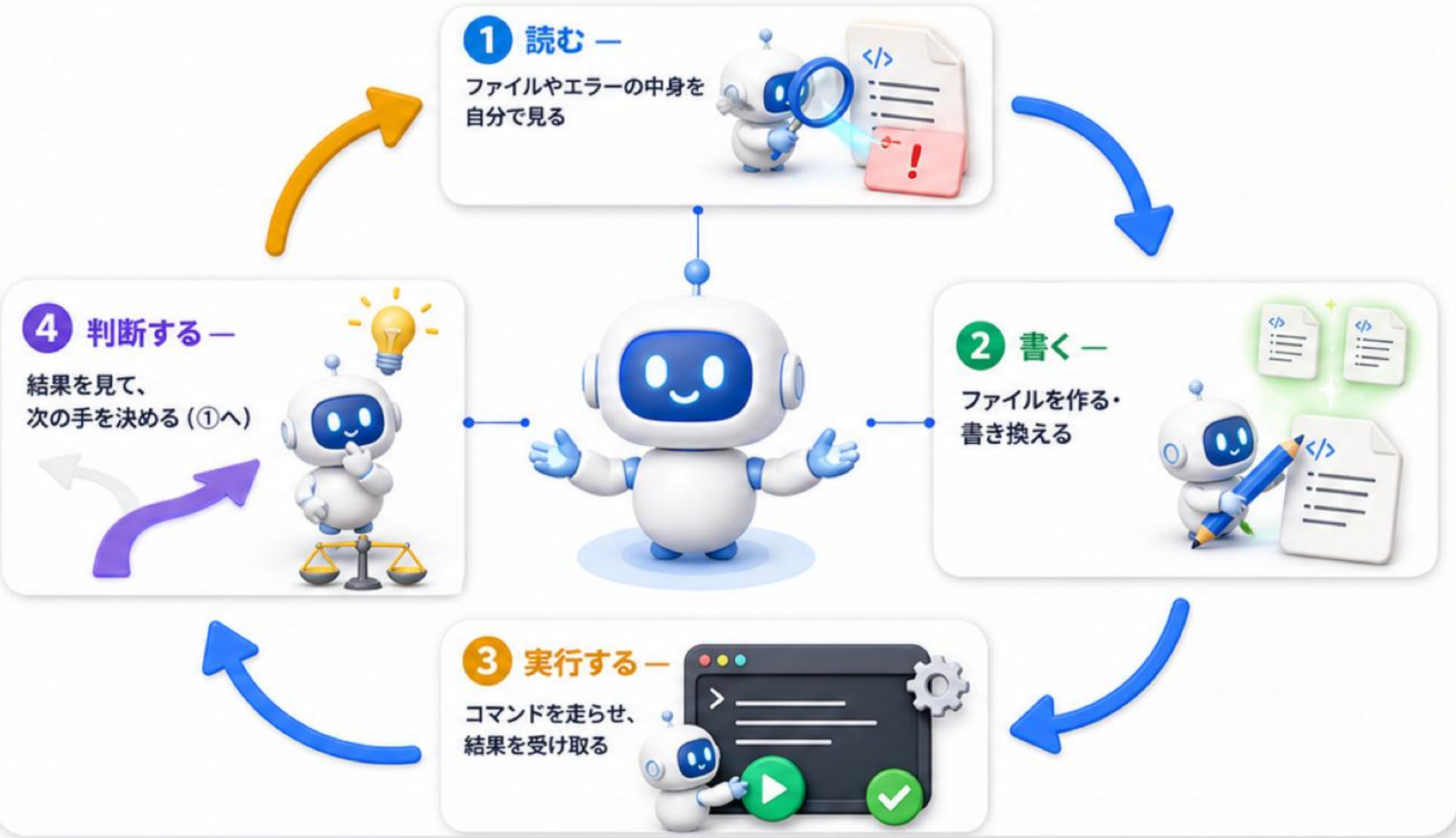
そして — 自分が何を作りたいのか、自分が分かるための文書でもある

コーディングエージェントの仕組み

チャット: 質問 → 回答（1往復）

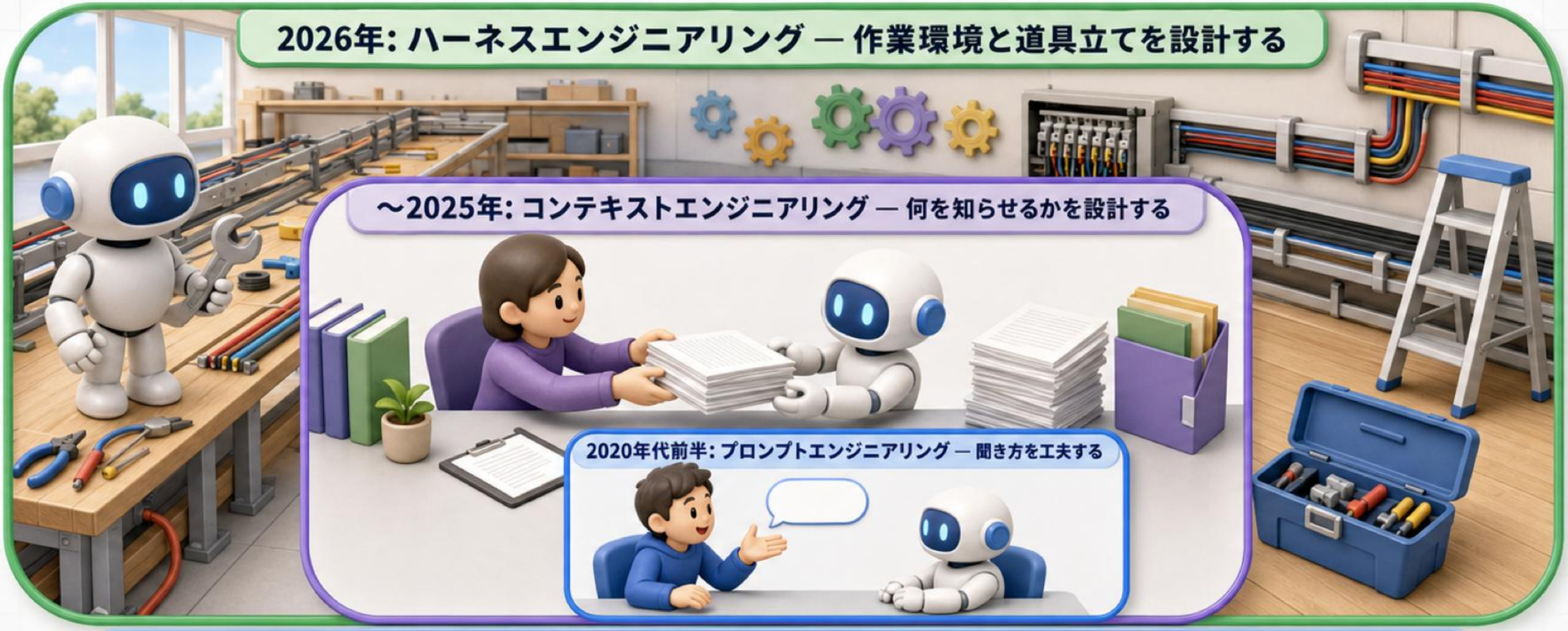


エージェント: 目標 → ツールを呼ぶループ



要点:
エージェントは「考えるだけ」ではなく「手を動かす」

ハーネスエンジニアリング



ハーネス = 指示 + 文脈 + 使える道具 + 実行環境 + 検証手段





入口はこの2ファイル：

AGENTS.md ／ .github/copilot-instructions.md

— プロジェクトの決まりごとを書いておくと、毎回言わなくて済む



Human in the loop / Human on the loop

in the loop — 一手ごとに人が確認して進める



例：変更の提案を1つずつ見て承認する。学習向き・失敗が小さい

on the loop — まとめて任せて、結果を検証する



例：30分放っておいて、出てきた差分をレビューする。速いが読む力が要る

今日のワークは in the loop から。自由制作で on the loop を試す

要点： 選ぶ基準は『失敗したとき何を失うか』

品質保証は、人間の仕事

ハルシネーションは減った。検証責任は減っていない



1 実行して確かめる

— 動くか。想定通りの答えを返すか



2 差分を読む

— 何が変わったのかを、自分の目で見る



3 テストを書かせる

— 確かめ方そのものをAIに作らせる



要点：動いたように見えるコードと、動くコードは別物

【ワーク1】数字当てゲーム：まず仕様書を書く

① プロジェクト直下に
SPEC.md を作る



SPEC.md

② 仕様を箇条書きで書く（下の5行でOK）

- 答えは1〜100の乱数
- ユーザーが数字を入力する
- 入力ごとに『もっと大きい』『もっと小さい』を表示する
- 当たるまで繰り返す
- 当たったら試行回数を表示して終了する



③ 余裕があれば1行足す
（例：数字以外が入力されたら注意する）

【ワーク1】Copilot agent mode に渡す



1 Copilot Chat を開き、モードを Agent にする

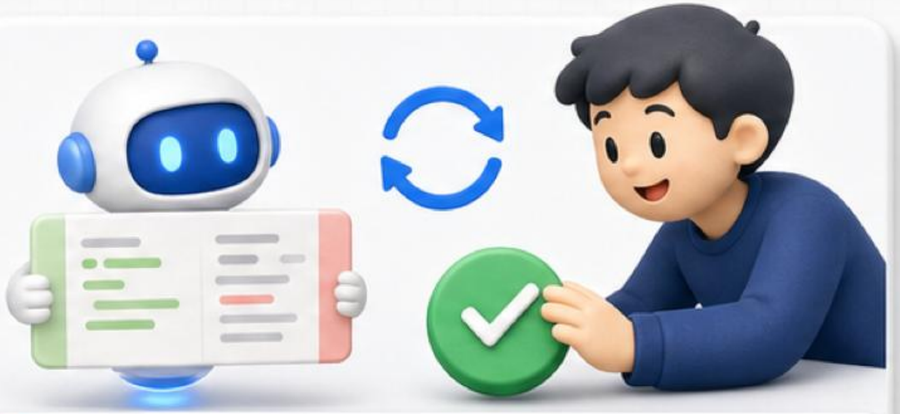


2 SPEC.md を明示的にコンテキストに追加する
（= エージェントに確実に見せる）



3 この依頼文を送る:

「SPEC.md の仕様に従って、guess.py を作成してください。
標準ライブラリだけを使ってください。」



4 提案された変更を見て、承認する ← Human in the loop

※ 実行を促されたら、ターミナルで: `uv run guess.py`

【ワーク1】実行して検証する



この5つを試す：



・正解より小さい数
→『もっと大きい』と出るか



・正解より大きい数
→『もっと小さい』と出るか



・1 と 100（境界）を入れてみる



・当てたとき、試行回数は
合っているか



・数字以外を入れたら、
どうなるか

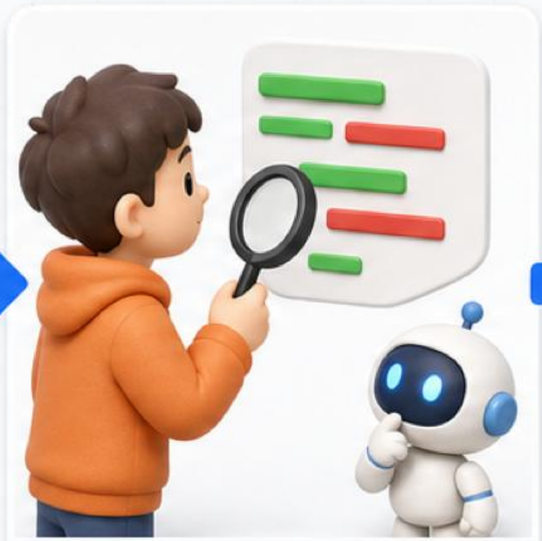


動かない・仕様と違う → それは失敗ではなく、次の1周の入力

【ワーク1】動いたらコミットする



VS Code のソース管理



差分を確認

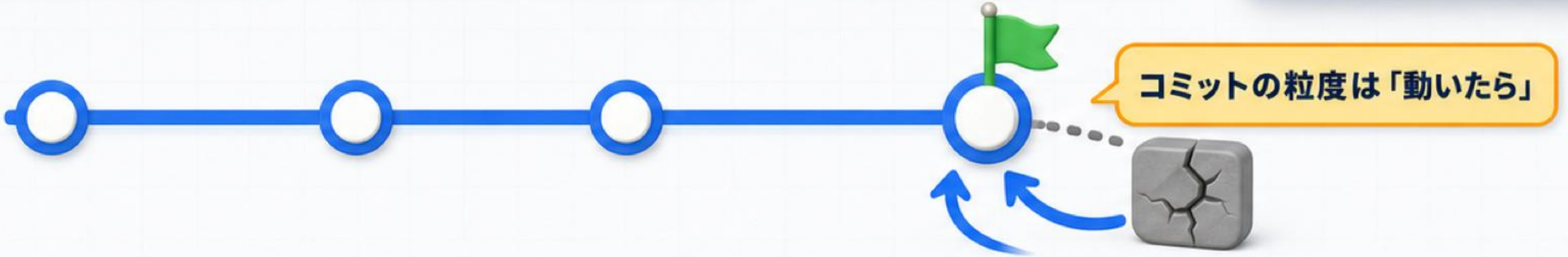


メッセージを書く



コミット

← ここまでが
今日の必須
到達点です



AIが次の1周で壊しても、ここに戻れる

うまくいかない時の3手



① エラーをそのまま貼る ー
要約しない。全文を貼る



② 仕様を直す ー
伝わっていないのは、たいてい仕様の側



③ 新しいセッションで仕切り直す ー
会話が長引いたら、捨てて始め直す

要点: 同じ指示を10回繰り返すより、仕様書を1行直す方が速い

【ワーク2】Streamlit で Web アプリにする

1 依存を追加する: uv add streamlit



2 SPEC.md に1行足す:「同じゲームを Streamlit の Web アプリにする」



3 agent mode に依頼する:



4 起動する: uv run streamlit run app.py



※ ブラウザが自動で開きます。止めるのは Ctrl+C

前の機能は、壊れていないか

新しい機能を足すと、古い機能が壊れることがある



Webアプリが動いた →
では、これも試す: uv run guess.py



これを リグレッション (regression) と呼ぶ



AIは頼まれていないファイルも
書き換えることがある



要点:
毎回ぜんぶ手で確かめるのは大変 → だからテストを書かせる

差分を見て、コミットする

① 差分で確認すること:



- app.py が増えた
- pyproject.toml に streamlit が追加された
- guess.py は（意図せず）変わっていないか

② 確認できたらコミット —

メッセージ例:『Streamlit版を追加』



2周目、完了 — これで型は身体に入りました

講師デモ: CLI エージェントの世界

Claude Code ／ Codex — ターミナルに住むコーディングエージェント



※ 今日は見るだけ。インストールは不要です

同じ課題を、CLI で流してみます



違い: 画面がコードではなくタスクになる



違い: 複数ファイルを横断し、自分でテストを実行する



違い: 数十分〜数時間、連続で作業を続けられる

長時間の自律作業と、メモリ管理

—— 長く働けるようになると、新しい問題が起きる ——



会話が長い → 文脈があふれる → 精度が落ちる

これがハーネス設計の中核

対策① 圧縮

これまでの経緯を
要約して持ち越す



対策② メモリファイル

決まりごとをファイルに書いて毎回読ませる
（AGENTS.md ／ CLAUDE.md）

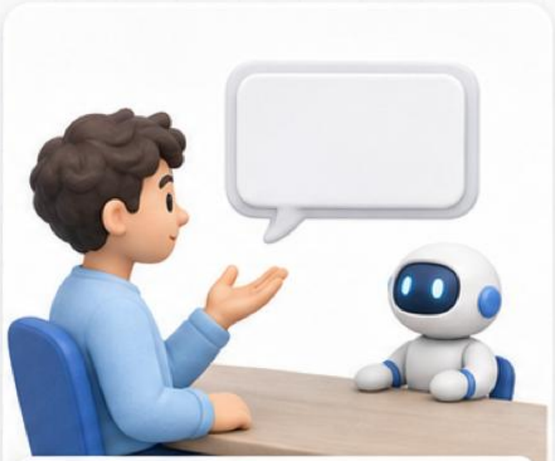


要点：皆さんの『新しいセッションで仕切り直す』も、同じ問題への対処

ツールは変わる。型は変わらない



2023年: コード補完



2024年: チャットで相談



2025年: エディタ内エージェント



2026年: 自律エージェント + ハーネス設計

この下に、ずっと同じものがある





要点: 道具の名前は覚えなくていい。型を覚えて帰ってください

自由制作：テーマの決め方

3つの入口：

1 **自分の困りごと ―**
毎週やっている面倒な作業

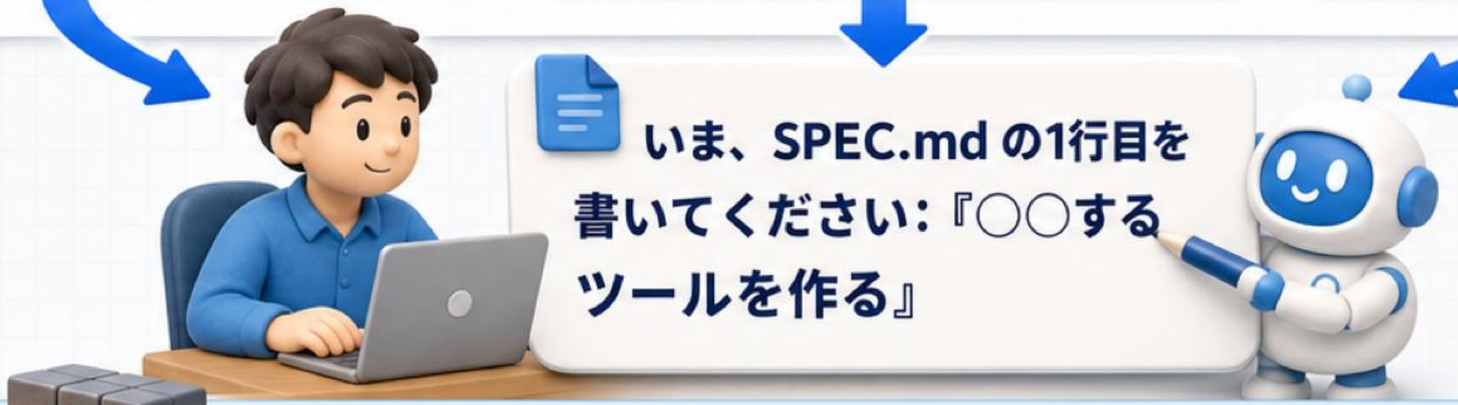


2 **研究テーマ ―**
ただし公開済みの内容・
公開情報か合成データのみ



3 **趣味 ―**
好きなものは、続く





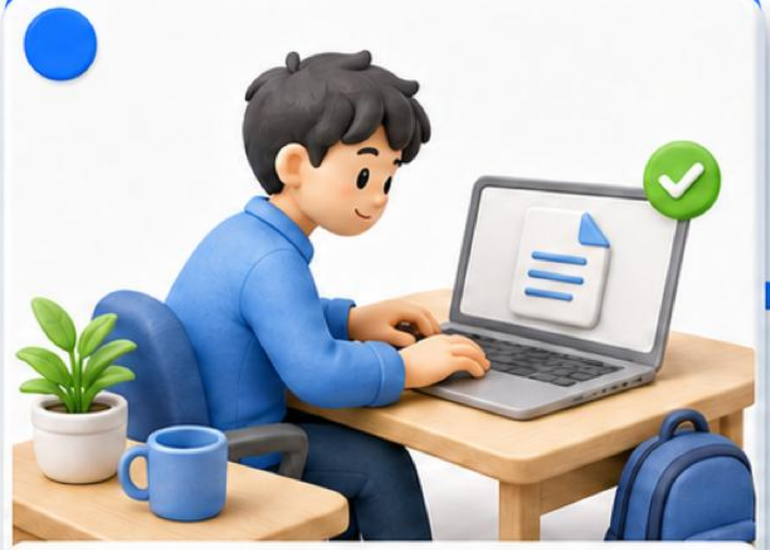


原則：小さく始める（明日の朝には動いているサイズに）



※ 未公開の研究データ・個人情報は使わない。リポジトリは private

制作の進め方と、今夜の使い方



いまここで：SPEC.md を書いて、最初のコミットまで



今夜（任意）：ループを1周だけ回してみる



明日の朝のゴール：「何を作るか」を一言で言える状態







詰まったら：エラーを貼る → 仕様を直す → 新しいセッション

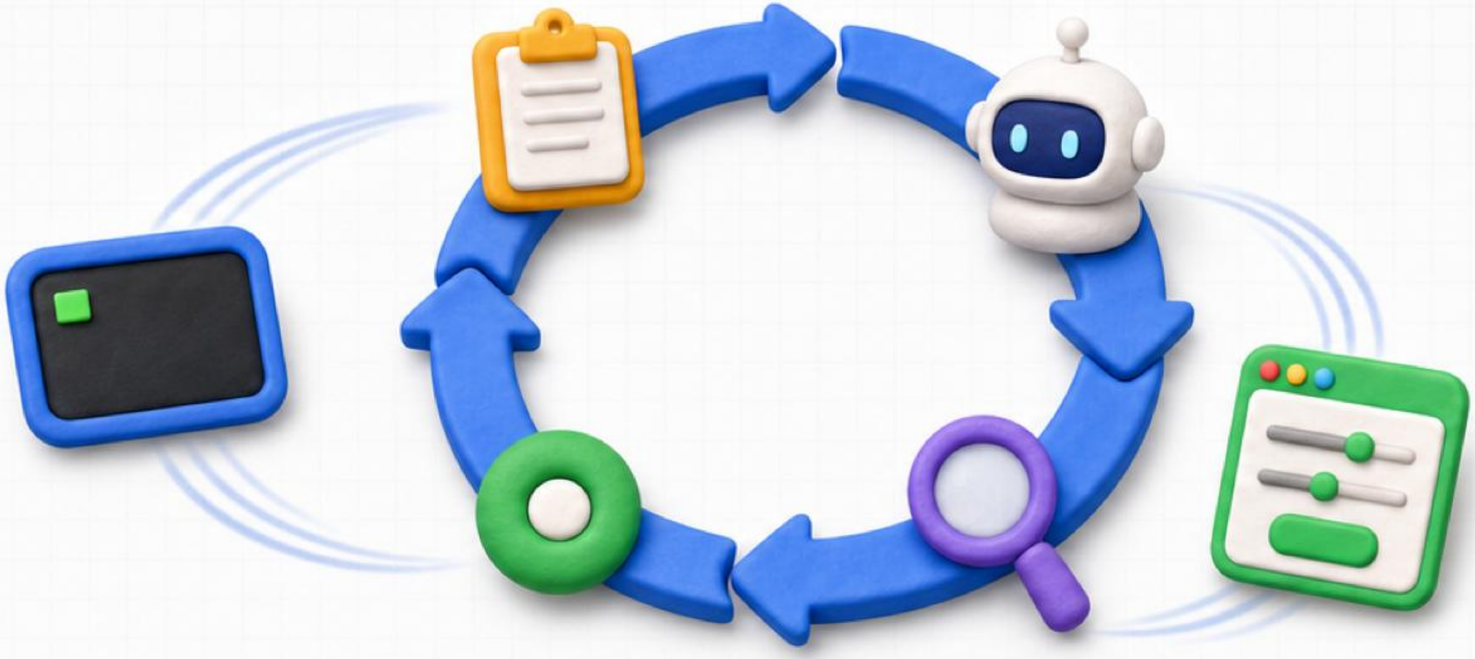


無理はしないでください。明日90分あります

※ 質問は明日の朝いちばんで受け付けます

1日目、完了 — 型は、もう手の中にある

今日やったこと: ループを2周





1 **数字当てゲーム (CLI) —**
仕様書 → 生成 → 検証 → コミット

2 **Streamlit版 —**
依存追加 → 生成 → 検証 → コミット

明日: 目標共有 (9:00) → 制作と push (9:15〜10:45) → LT (11:00)