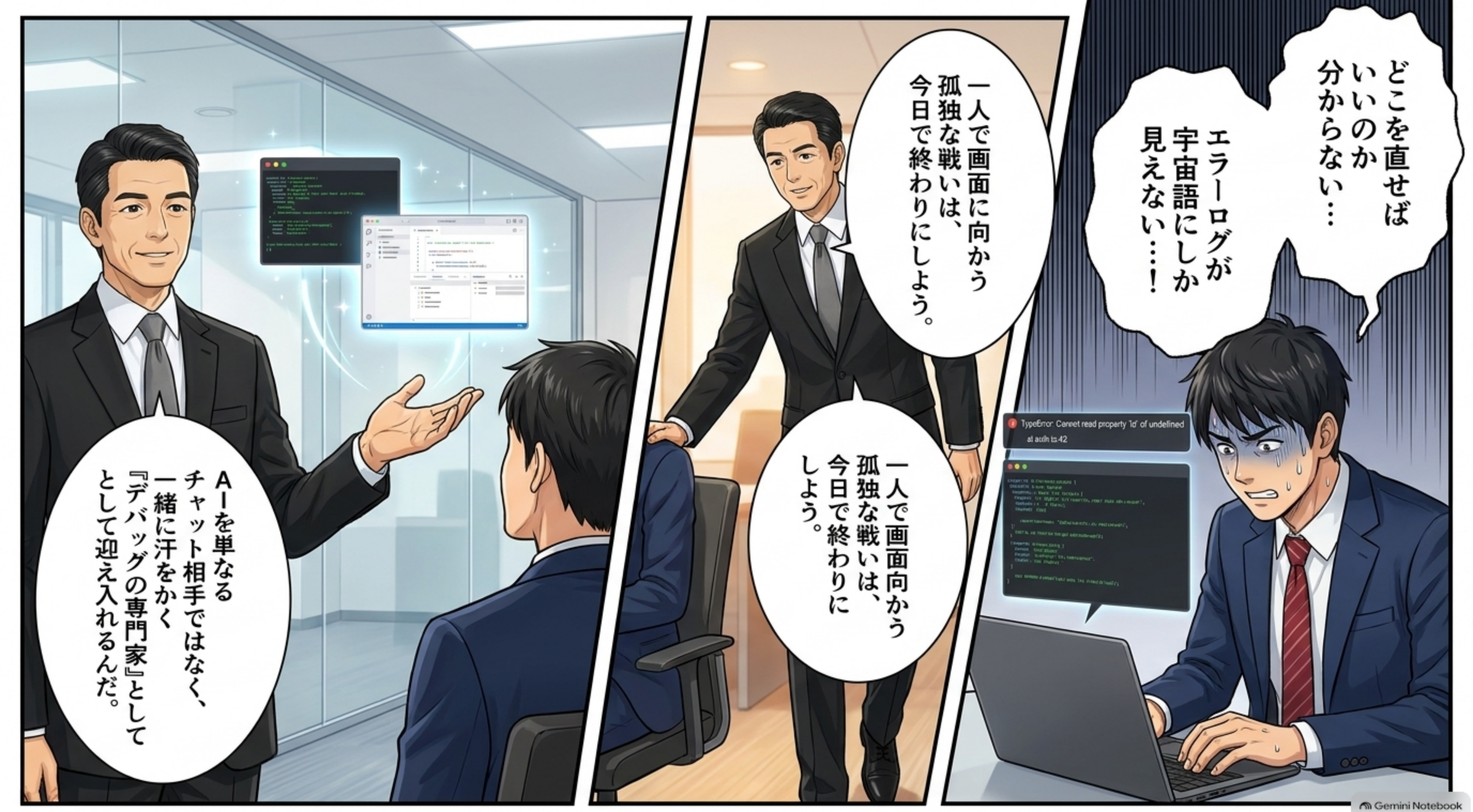
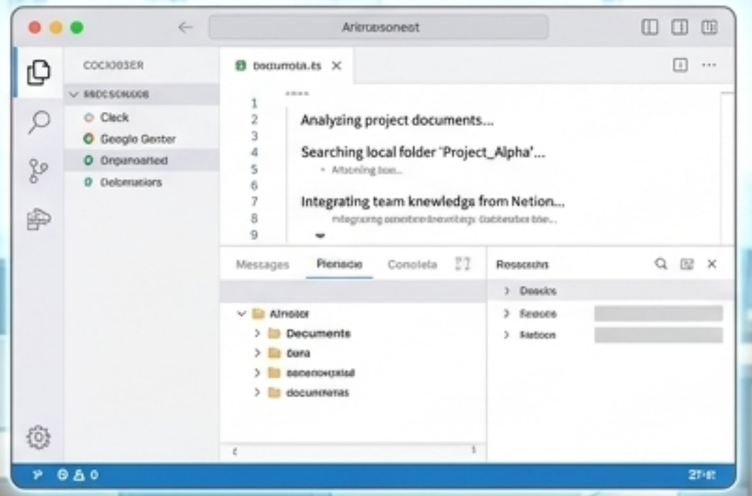

バグ調査の孤独な戦いを終わらせる
AIデバッグ指揮官への道
Terminal
Claude Code & Cowork
実践ガイド
Gemini Notebook

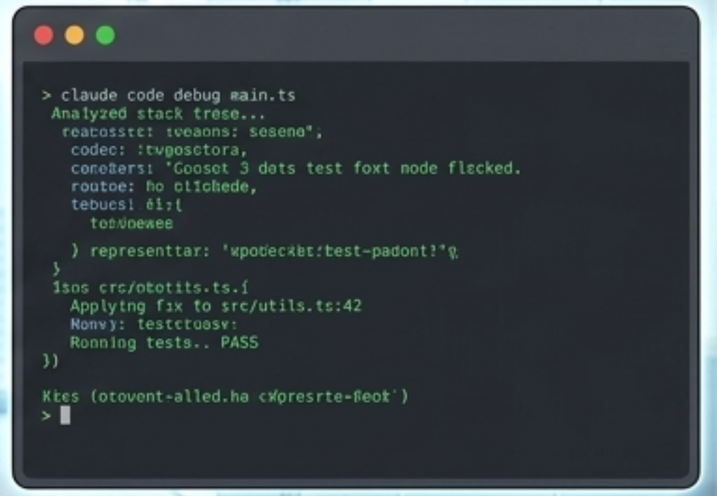

どこを直せば
いいのか
分からない…
エラーログが
宇宙語にしか
見えない…！
TypeError: Cannot read property 'id' of undefined
at auth.ts:42
一人で画面に向かう
孤独な戦いは、
今日で終わりにしよう。
一人で画面向かう
孤独な戦いは、
今日で終わりに
しよう。
AIを単なる
チャット相手ではなく、
一緒に汗をかく
『デバッグの専門家』として
として迎え入れるんだ。

【有能な秘書：Claude Cowork】

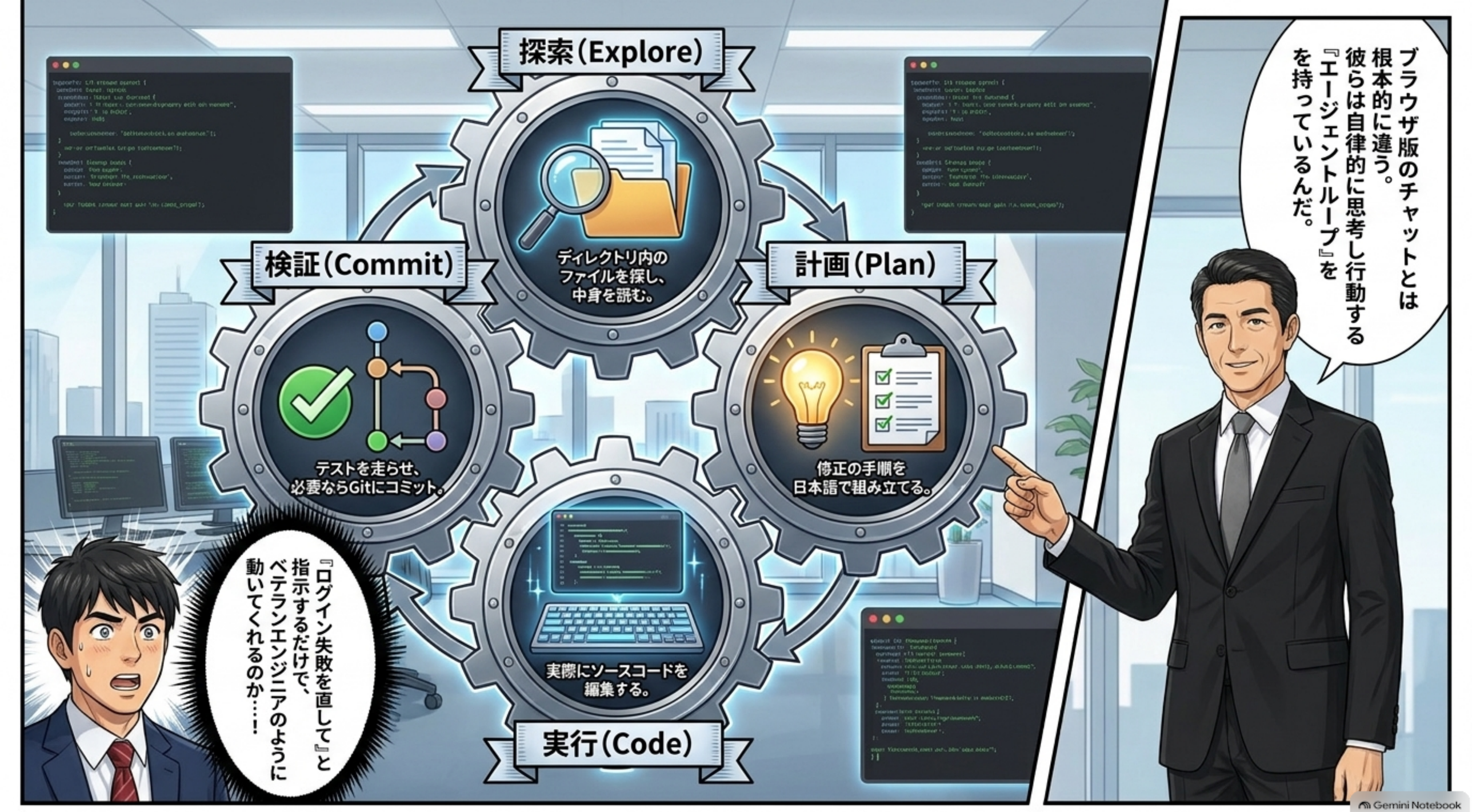


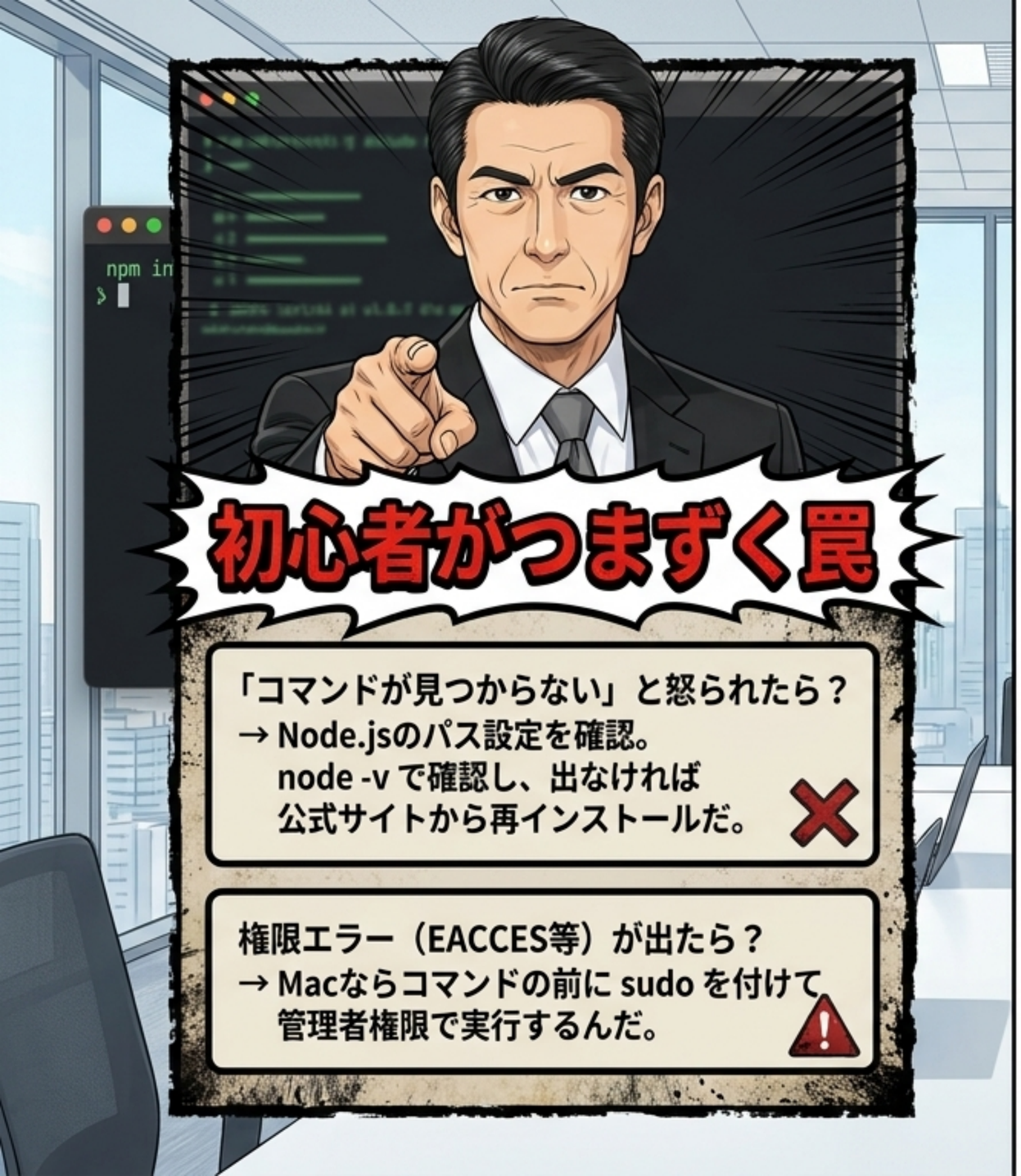
- ブラウザ/デスクトップ (GUI)
- 業務ツールやローカルフォルダを横断
- チームのナレッジを統合して支援

【実務家：Claude Code】



- ターミナル専用 (CUI)
- エンジニア特化型
- ソースコードを直接読み、書き換え、テストを実行

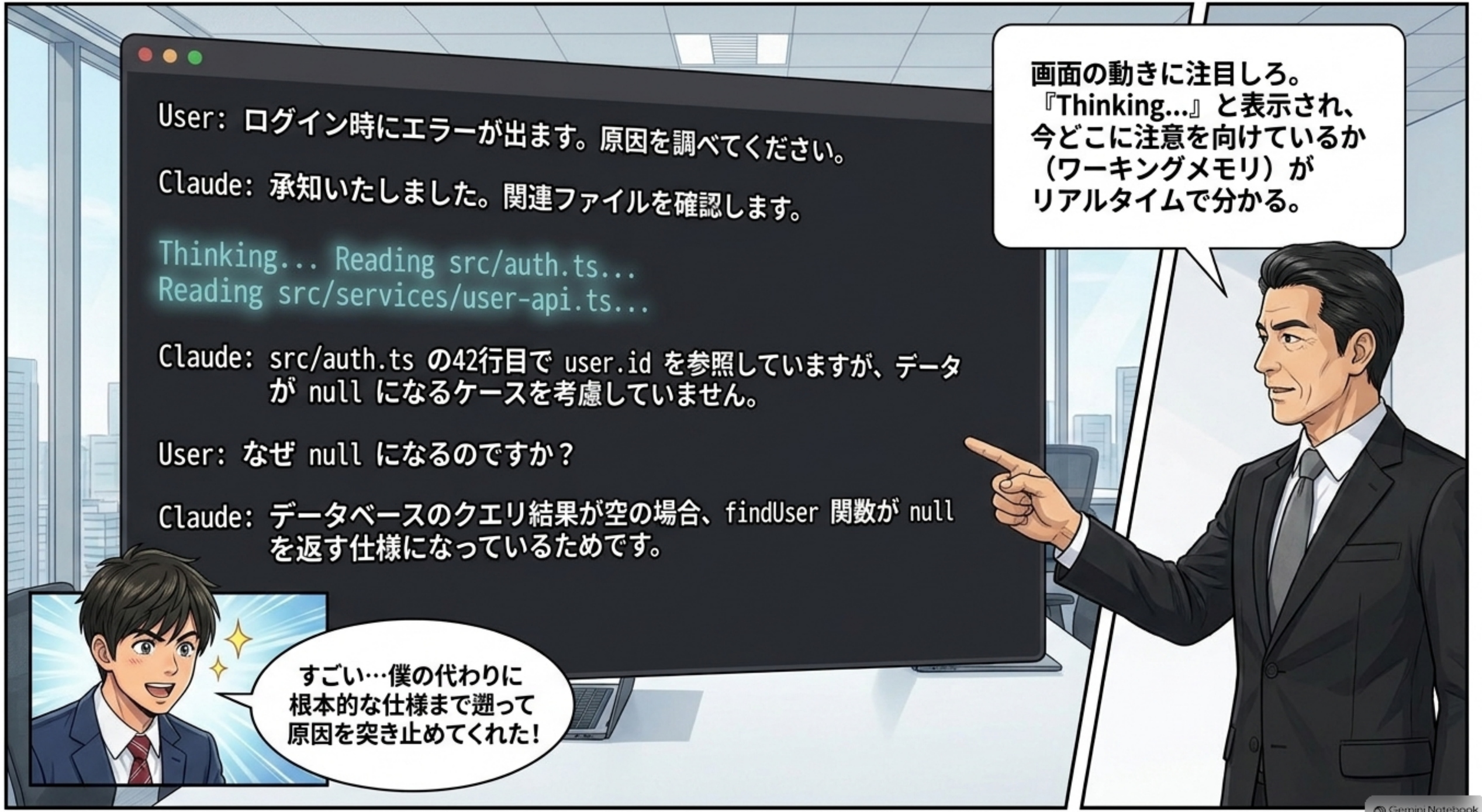

探索（Explore）
ディレクトリ内のファイルを探し、中身を読む。
計画（Plan）
修正の手順を日本語で組み立てる。
実行（Code）
実際にソースコードを編集する。
検証（Commit）
テストを走らせ、必要ならGitにコミット。
ブラウザ版のチャットとは根本的に違う。彼らは自律的に思考し行動する『エージェントループ』を持っているんだ。
『ログイン失敗を直して』と指示するだけで、ベテランエンジニアのように動いてくれるのか……！



```
<error_log>
TypeError: Cannot read property
  'id' of undefined
  at auth.ts:42
</error_log>

<goal>
上記のエラーの原因を特定し、修正のための計画を立ててください。
</goal>
```

タグはClaudeへの
「ここがエラーの情報だよ」
という明確な目印になる。


User: ログイン時にエラーが出ます。原因を調べてください。
Claude: 承知いたしました。関連ファイルを確認します。
Thinking... Reading src/auth.ts...
Reading src/services/user-api.ts...
Claude: src/auth.ts の42行目で user.id を参照していますが、データが null になるケースを考慮していません。
User: なぜ null になるのですか？
Claude: データベースのクエリ結果が空の場合、findUser 関数が null を返す仕様になっているためです。
画面の動きに注目しろ。
『Thinking...』と表示され、
今どこに注意を向けているか
（ワーキングメモリ）が
リアルタイムで分かる。
すごい…僕の代わりに
根本的な仕様まで遡って
原因を突き止めてくれた！

アーティファクト効果の罠

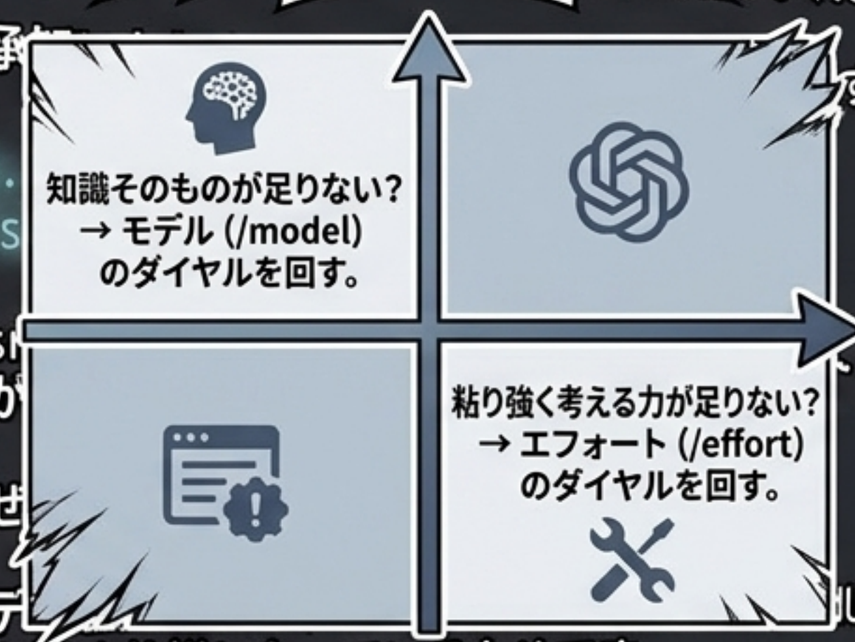
AIが書いたコードは見た目が非常に綺麗（アーティファクト）なため、人間は「正しいに違いない」と思い込み、見極め（Discernment）をサボる傾向がある。

（※評価行動が約3〜5%低下する研究データあり）

幻覚（ハルシネーション）を見抜くための3つの質問：

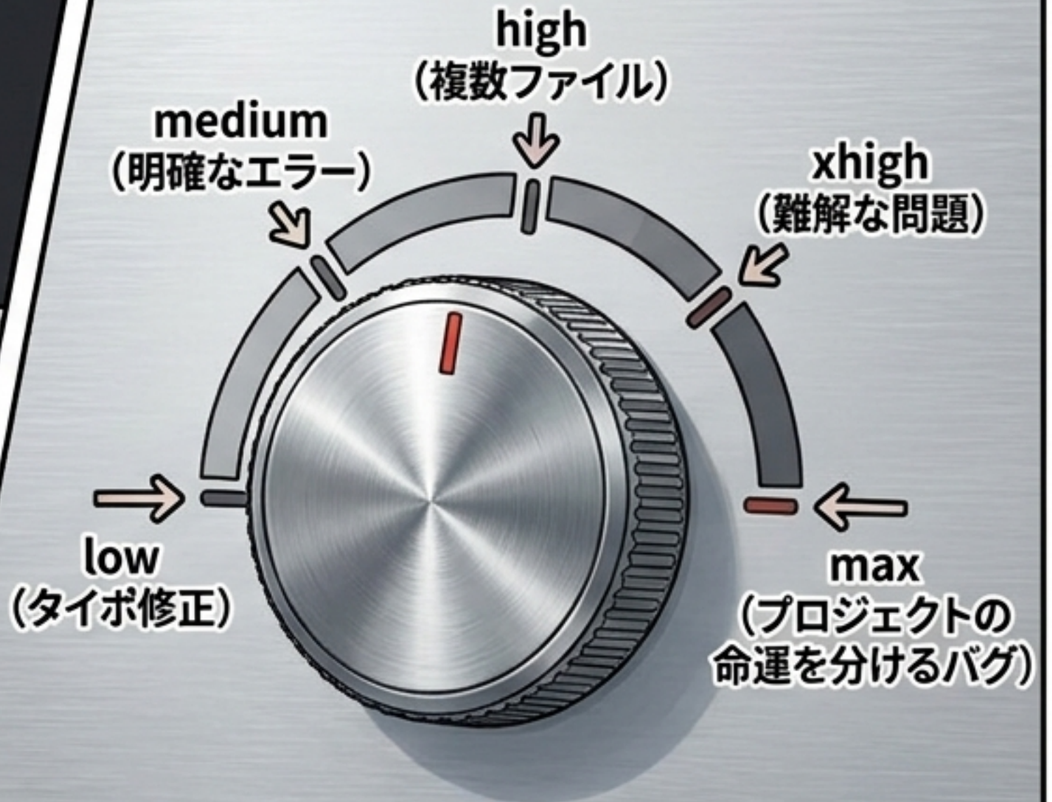
1. 「この修正による副作用（悪影響）はありますか？」
2. 「前提としている条件を3つ教えてください。」
3. 「あえてこの修正に反対する理由は何ですか？」

うまくいかない時の診断チャート



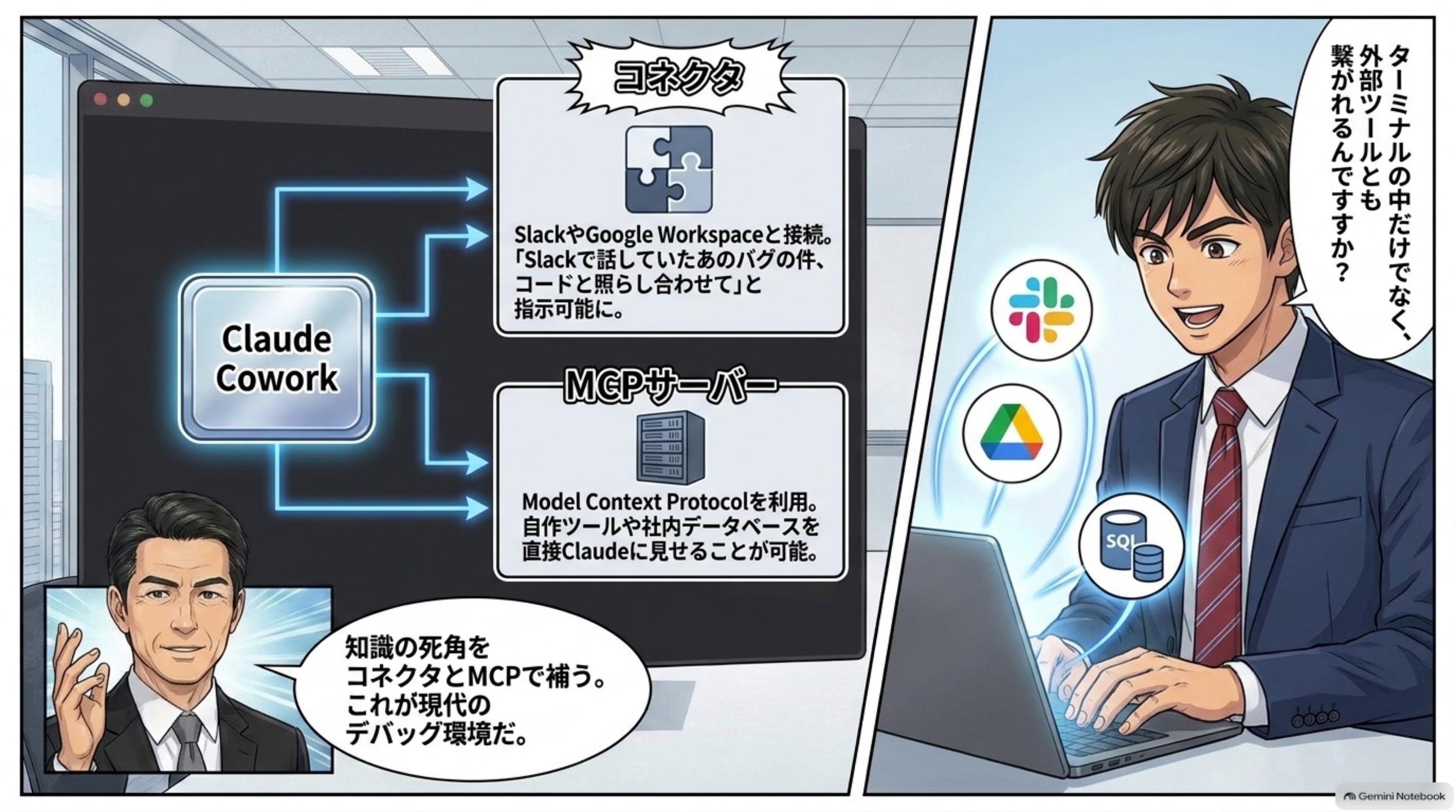
定石は『まずエフォートを1段階上げて再挑戦』だ。/effort max の自己反論は必見だぞ。

/effort



バグの難易度に応じて、Claudeに割かせるリソース（頑張り度合い）を5段階で調整できる。


CLAUDE.md の掟（記述例）
- 命名規則: 変数名は必ずキャメルケース（exampleVariable）に。
- 命名規則: 変数名は必ずキャメルケース（exampleVariable）に。
- デバッグルール: console.log は禁止。logger.debug を使用すること。
- エラー処理: DBエラーは CustomDatabaseException でラップする。
- 重要: 以前、IDが null になるバグがあったため、ユーザーデータの参照前には必ず存在確認を行うこと。
CLAUDE.md
これを置くだけで、Claudeは「このプロジェクトの流儀」に沿った提案をするようになる。
直った…！ でも、また同じようなバグがバグが起きたら面倒ですね。
未来のバグを防ぐ『プロジェクトメモリ（永続的な記憶）』を育てるんだ。ルートディレクトリにこのファイルを置け。


コネクタ
SlackやGoogle Workspaceと接続。
「Slackで話していたあのバグの件、
コードと照らし合わせて」と
指示可能に。
Claude
Cowork
MCPサーバー
Model Context Protocolを利用。
自作ツールや社内データベースを
直接Claudeに見せることが可能。
知識の死角を
コネクタとMCPで補う。
これが現代の
デバッグ環境だ。
ターミナルの中だけでなく、
外部ツールとも
繋がれるんですか？
SQL

あなたも今日から「デバッグの指揮官」

面倒な探索は彼らに任せ、君は『どんな価値をユーザーに届けたいか』に集中しろ。

今日試すべき「最初のアクション」

1. 一番小さなエラーログをコピーする。
2. Claudeにこう話しかける：

「このエラーに悩まされているんだけど、解決のために何が起きているか一緒に考えてくれる？」

【作業員】
コードを一文字ずつ直す孤独な作業。

【指揮官】
AIチームに指示を出し、最終判断を下す。

さあ、ターミナルを開いて、新しいデバッグの旅に出かけよう！