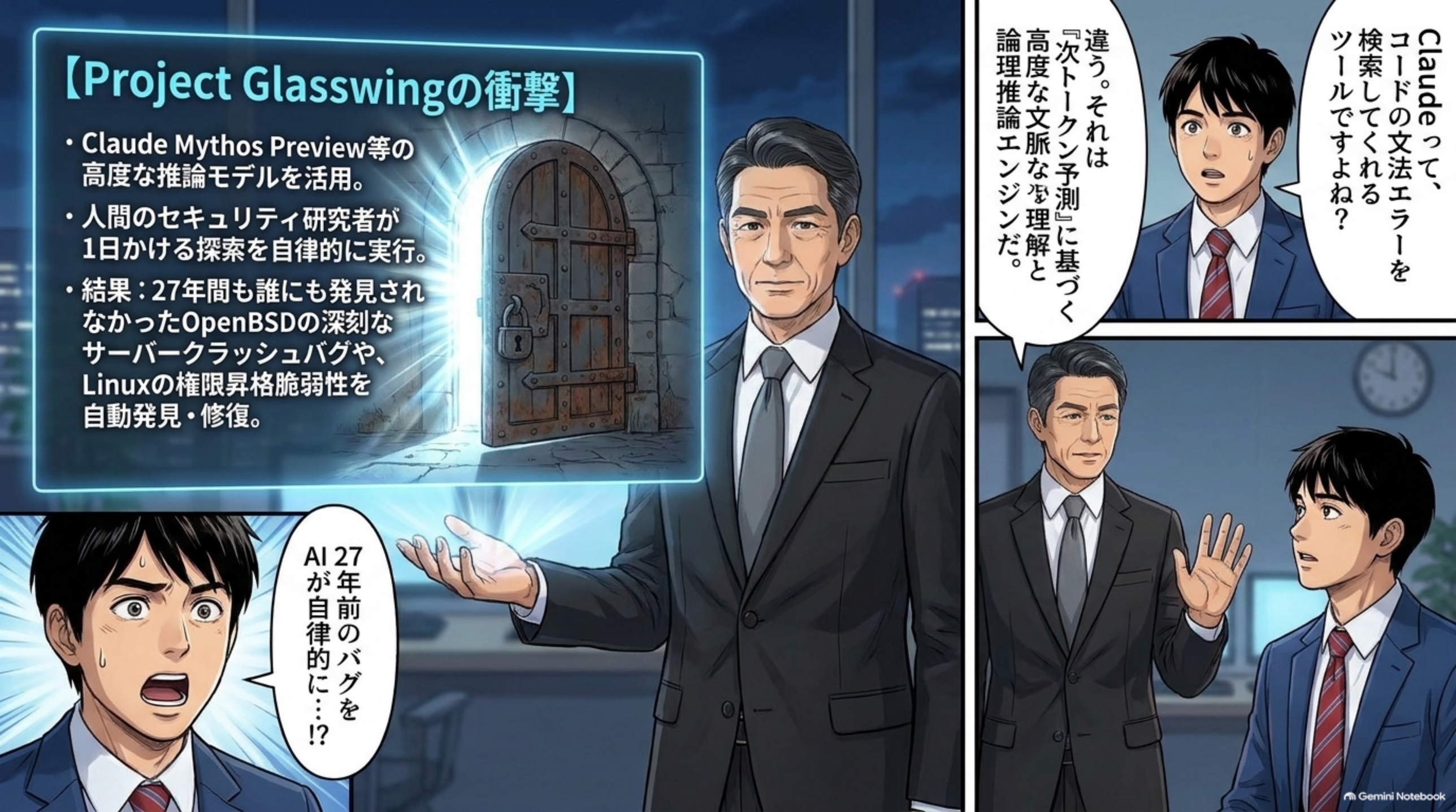

AIにデバッグを
任せる第一歩
Claude完全攻略ガイド
原因不明のエラーに沼る日々は、今日で終わる。
AIは『検索エンジン』ではなく、一緒に考える『最強の相棒』だ！


開発や研究の現場では、
全作業時間の約80%が
デバッグや調整に費やされて
と言われている。
本来注力すべきロジック構築に
ロジック構築にないと
辿り着けていないんじゃいか？
はい……でも、
どうすれば……
やり方を根本から変えよう。
AIツール『Claude』を
開発パートナーとして迎え
迎え入れるんだ。
まただ…どこに潜んでいるかわからない
１行の記述ミスのせいで、
今日も１日が終わってしまった……
Gemini Notebook


【Project Glasswingの衝撃】
・Claude Mythos Preview等の高度な推論モデルを活用。
・人間のセキュリティ研究者が1日かける探索を自律的に実行。
・結果：27年間も誰にも発見されなかったOpenBSDの深刻なサーバークラッシュバグや、Linuxの権限昇格脆弱性を自動発見・修復。
AIが自律的に…!?
27年前のバグを
Claudeって、コードの文法エラーを検索してくれるツールですよね？
違う。それは『次トークン予測』に基づく高度な文脈理解と論理推論エンジンだ。
Gemini Notebook

AIとの協働を成功させる
4D AI Fluency



Delegation
(委任力):
適切な粒度で
AIに任せる



Description
(記述力):
背景と意図を
正確に言語化
する



Discernment
(見極め・評価力):
結果を鵜呑みに
せず批判的に
吟味する

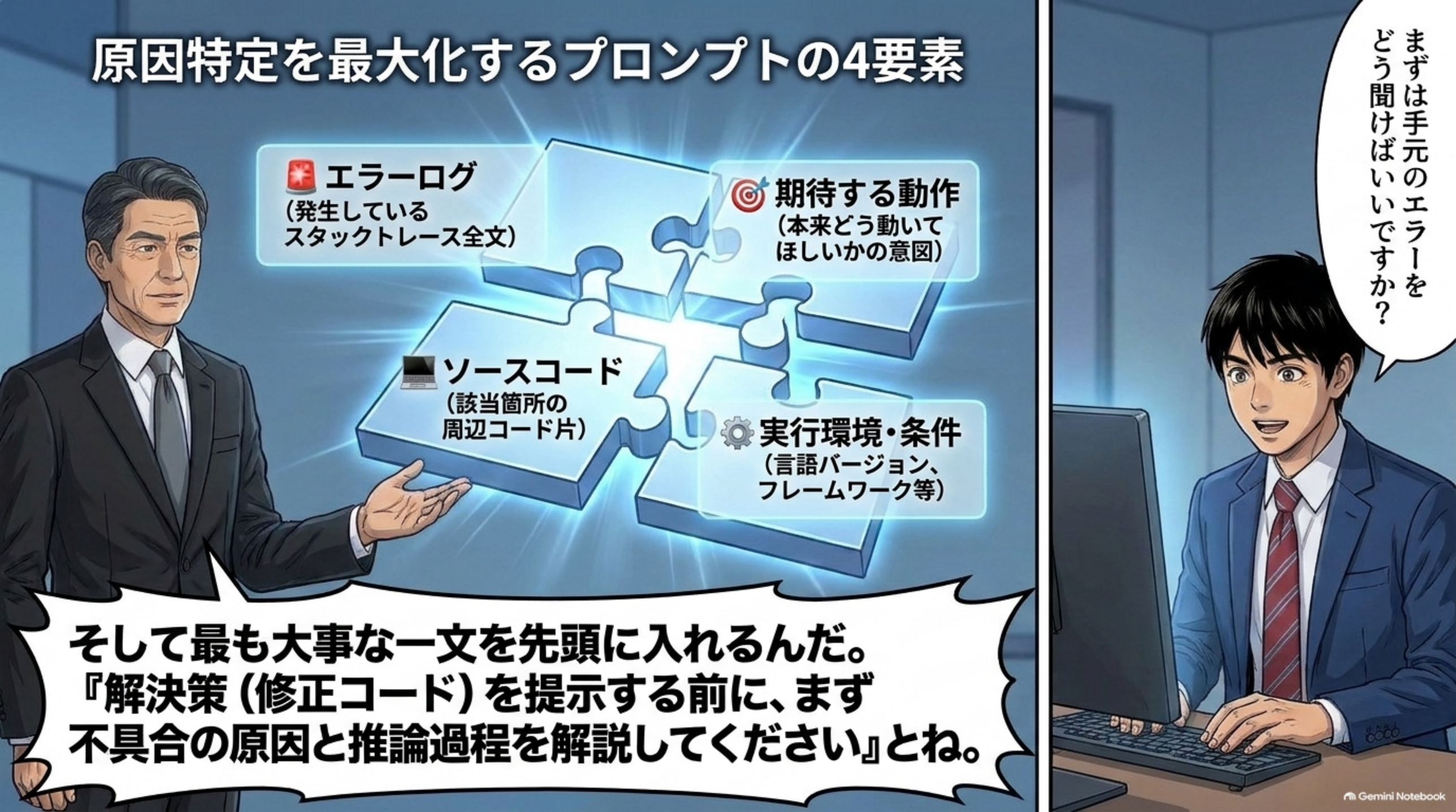


Dilig
(倫理的責任・誠実さ):
最終的なコード
品質の責任は
人間が負う

AIにすべてを
丸投げしてはいけない。

君が『指揮者』と
適切な指示
(Description)を
を与え、
結果を見極め(Discernment)、
責任を担う(Diligence)んだ。
んだ。

Gemini Notebook


原因特定を最大化するプロンプトの4要素
エラーログ
(発生しているスタックトレース全文)
期待する動作
(本来どう動いてほしいかの意図)
ソースコード
(該当箇所の周辺コード片)
実行環境・条件
(言語バージョン、フレームワーク等)
そして最も大事な一文を先頭に入れるんだ。
『解決策（修正コード）を提示する前に、まず
不具合の原因と推論過程を解説してください』とね。
まずは手元のエラーをどう聞けばいいですか？
Gemini Notebook

TypeError: unsupported operand type(s) for +: 'int' and 'str'

```
total += user_data.get('bonus_score', '0')
```

base_score は数値 100

get('bonus_score', '0') で取得した値が文字列 '50' として渡された

100 + '50' という「数値＋文字列」の計算が実行され型不整合が発生



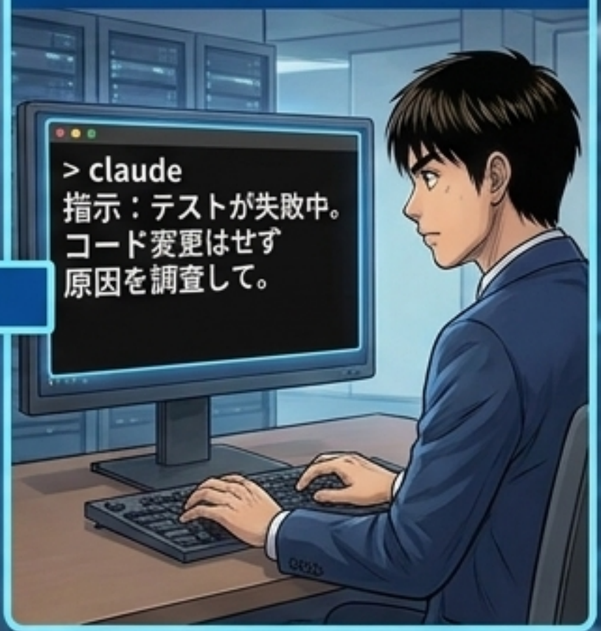
Gemini Notebook


アーティファクト効果
（The AI Fluency Indexより）
綺麗な成果物を見たとき：
・推論への疑問行動 -3.1pt
・事実確認行動 -3.7pt
・コンテキスト特定 -5.2pt
待て！！
（Discernmentを
忘れるな）
綺麗なコード！
これで一発解決ですね、
コミットします！
見た目が綺麗だと人間の検証能力は落ちる。
一発の回答で満足せず、
必ず反復（Iteration）して問い直せ。
反復することで『推論への疑問』は
5.6倍に跳ね上がる。
Gemini Notebook

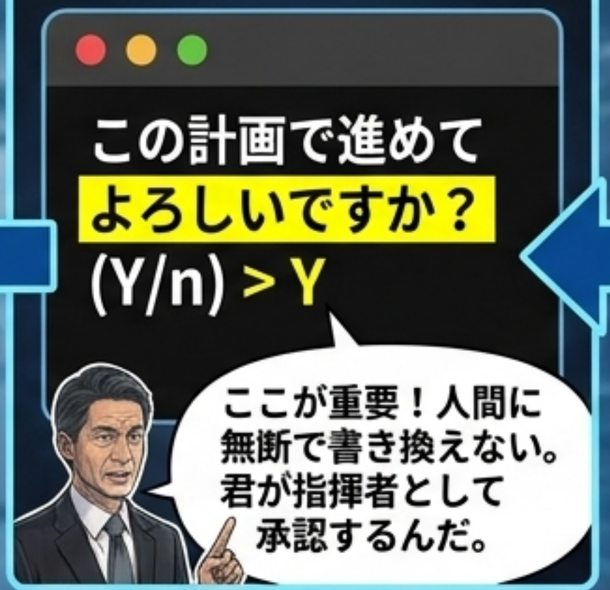

点のデバッグは理解したな。
次は『面』だ。
ターミナル上で自律的に動くAI
コーディングエージェント
『Claude Code』を起動する。
🔍 コンテキスト収集
(Read / Search / Grep)
- ファイル検索、エラー箇所特定
Agentic
Loop
✅ 結果の検証
(Verify & Iterate)
- テスト確認、失敗なら
1へ戻る
⚙ アクション実行
(Plan / Edit / Bash)
- 修正計画、コード書き
換え、テスト実行
AIが自分で
ファイルを横断して、
テストまで自動で
回してくれるんで
ですか!?

Claude Code 実践ワークフロー：4つのステップ

Step 1: Explore（探索）🔍



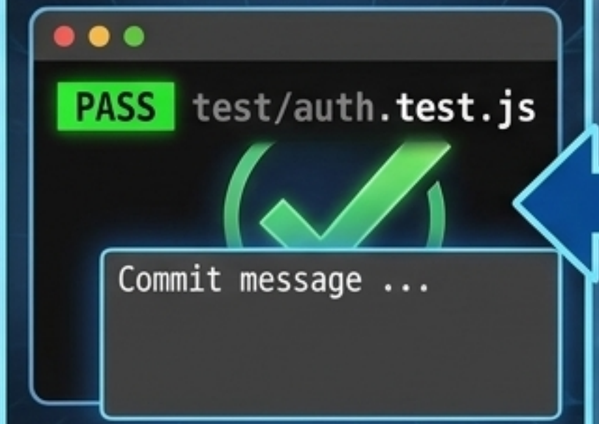
Step 2: Plan（計画承認）📝



Step 3: Code（修正）💻

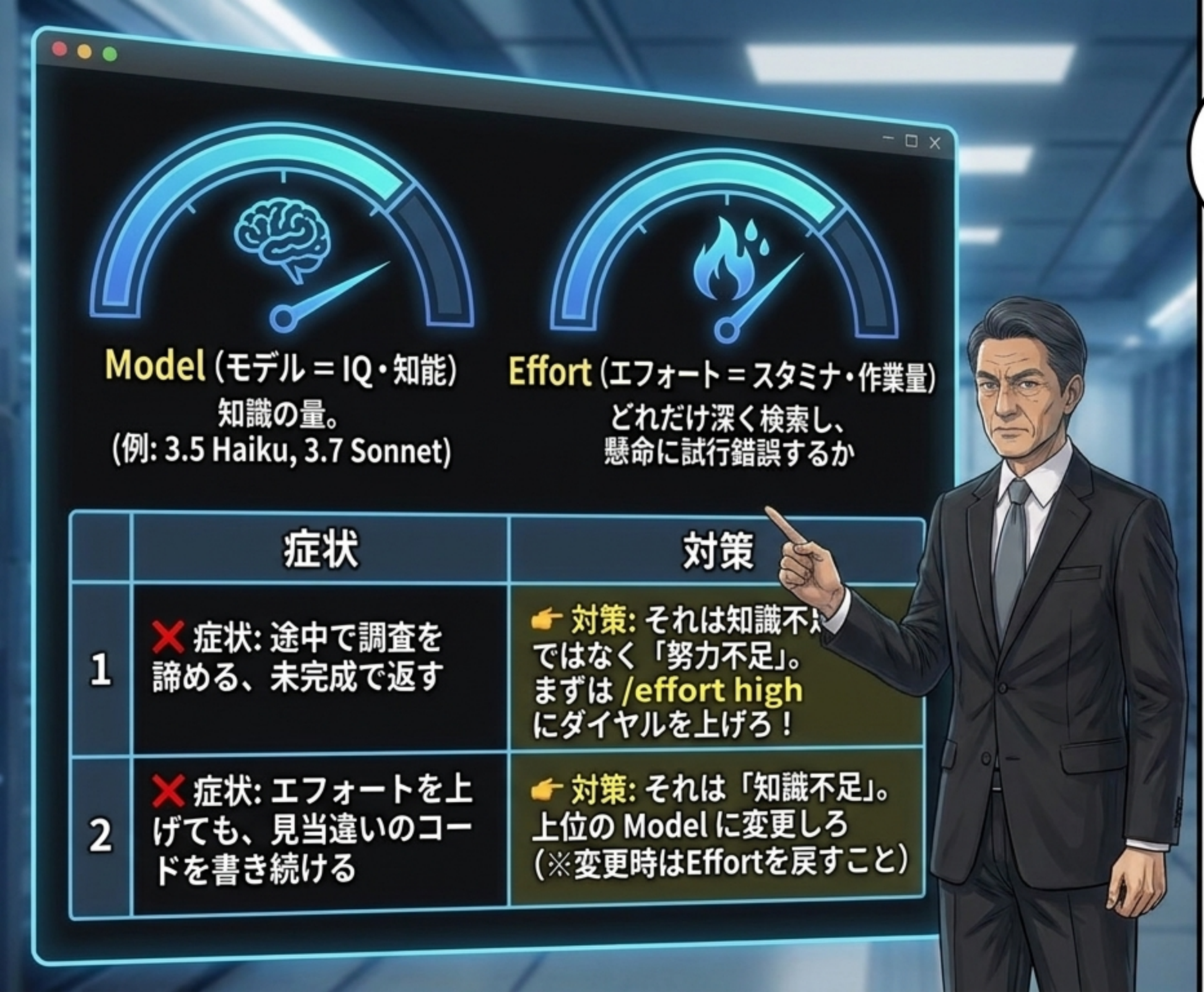


Step 4: Commit（完了）✅

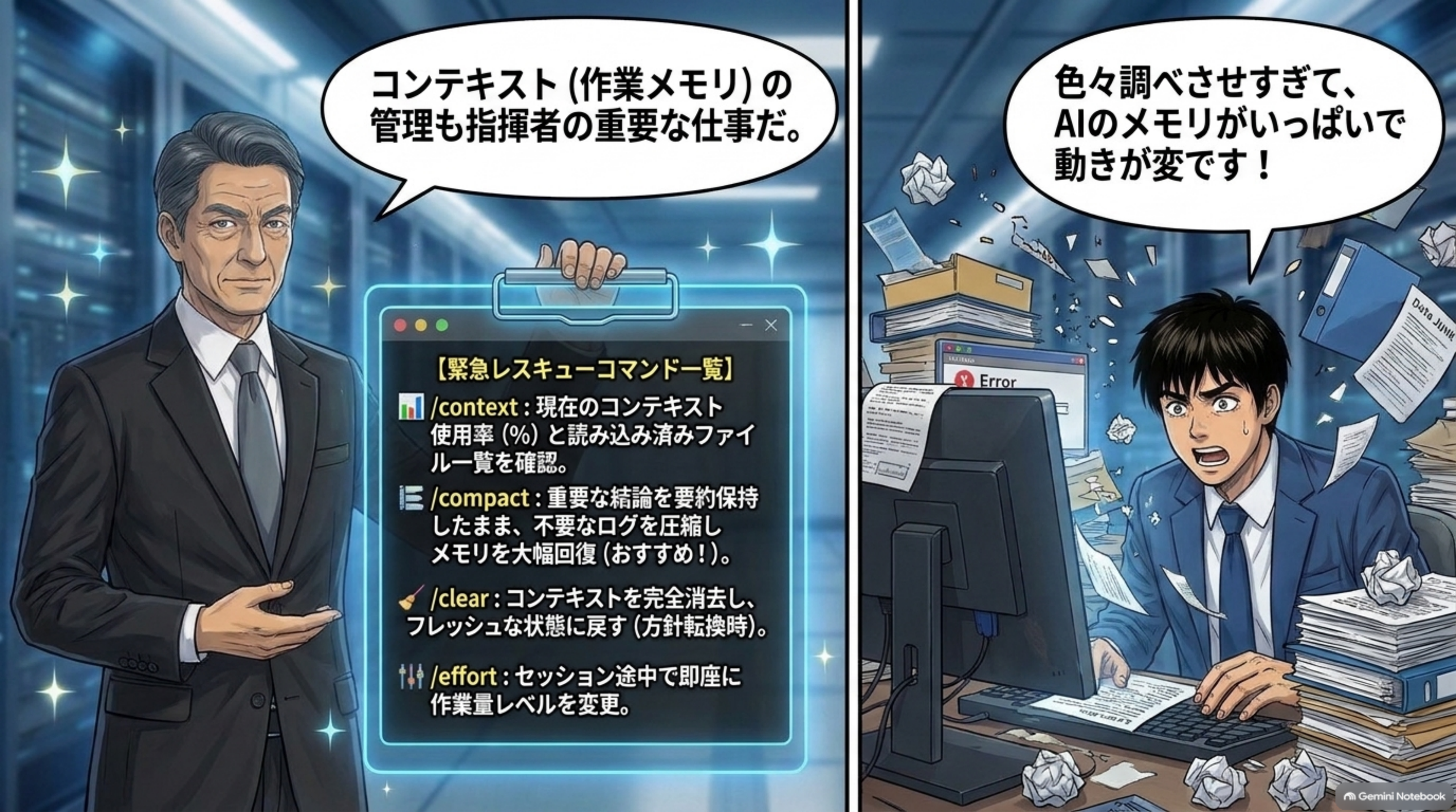


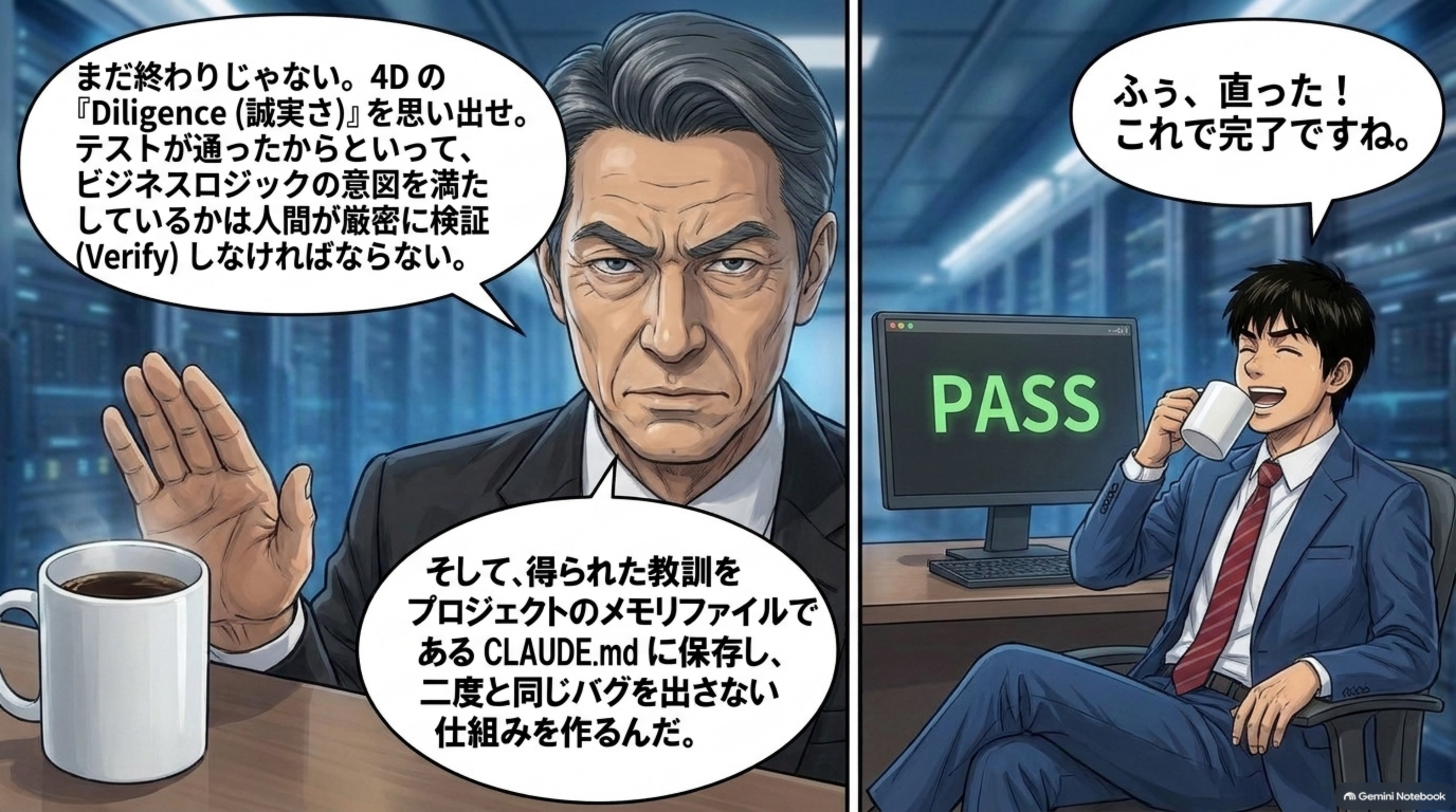
なるほど…全行手動で追う必要はない。
僕は『ディレクター』として計画を監修すればいいんだ！

Gemini Notebook


Model（モデル＝IQ・知能）
知識の量。
(例: 3.5 Haiku, 3.7 Sonnet)
Effort（エフォート＝スタミナ・作業量）
どれだけ深く検索し、
懸命に試行錯誤するか
症状
対策
1
❌症状: 途中で調査を諦める、未完成で返す
👉対策: それは知識不
ではなく「努力不足」。
まずは /effort high
にダイヤルを上げろ！
2
❌症状: エフォートを上げても、見当違いのコードを書き続ける
👉対策: それは「知識不足」。
上位の Model に変更しろ
（※変更時はEffortを戻すこと）


あれ…途中でAIが調査を
諦めちゃいました。
テストの修正も中途半端です。
Gemini Notebook


コンテキスト（作業メモリ）の
管理も指揮者の重要な仕事だ。
【緊急レスキューコマンド一覧】
/context：現在のコンテキスト使用率（%）と読み込み済みファイル一覧を確認。
/compact：重要な結論を要約保持したまま、不要なログを圧縮しメモリを大幅回復（おすすめ！）。
/clear：コンテキストを完全消去し、フレッシュな状態に戻す（方針転換時）。
/effort：セッション途中で即座に作業量レベルを変更。
色々調べさせすぎて、
AIのメモリがいっぱいで
動きが変です！
Error
Data JUNK
Gemini Notebook


まだ終わりじゃない。4D の『Diligence (誠実さ)』を思い出せ。テストが通ったからといって、ビジネスロジックの意図を満たしているかは人間が厳密に検証 (Verify) しなければならない。
そして、得られた教訓をプロジェクトのメモリファイルである CLAUDE.md に保存し、二度と同じバグを出さない仕組みを作るんだ。
ふぅ、直った！これで完了ですね。
PASS
Gemini Notebook

○ 書くべきこと：プロジェクト固有のルール

- 日時処理の内部タイムスタンプは例外なくUTC固定
- クライアントでのローカルタイム変換メソッド使用禁止

（※具体的で検証可能なルールのみを箇条書きにする）

CLAUDE.md

プロジェクト開発
ガイドライン＆
ナレッジベース

× 書かないこと：一般論による肥大化

- Pythonのif文の書き方
- 綺麗にコードを書くこと

（※コンテキストウィンドウを無駄に消費し、精度低下を招くため）



個人のデバッグ経験が、
AIを通じてチーム全体の
『永続的な資産』に
変わるんですね！



Gemini Notebook


デバッグはストレスに満ちた泥沼の捜索から、
AIとともに解き明かす『知的なパズル』へと変わった。
さあ、恐れることはありません。新しい開発体験を始めましょう。
👉 Action: 今すぐ、直近で発生したエラーログをコピーして、Claudeにこう入力してみてください:
『解決策を出す前に、このエラーの原因と推論過程を解説して』と。
Gemini Notebook