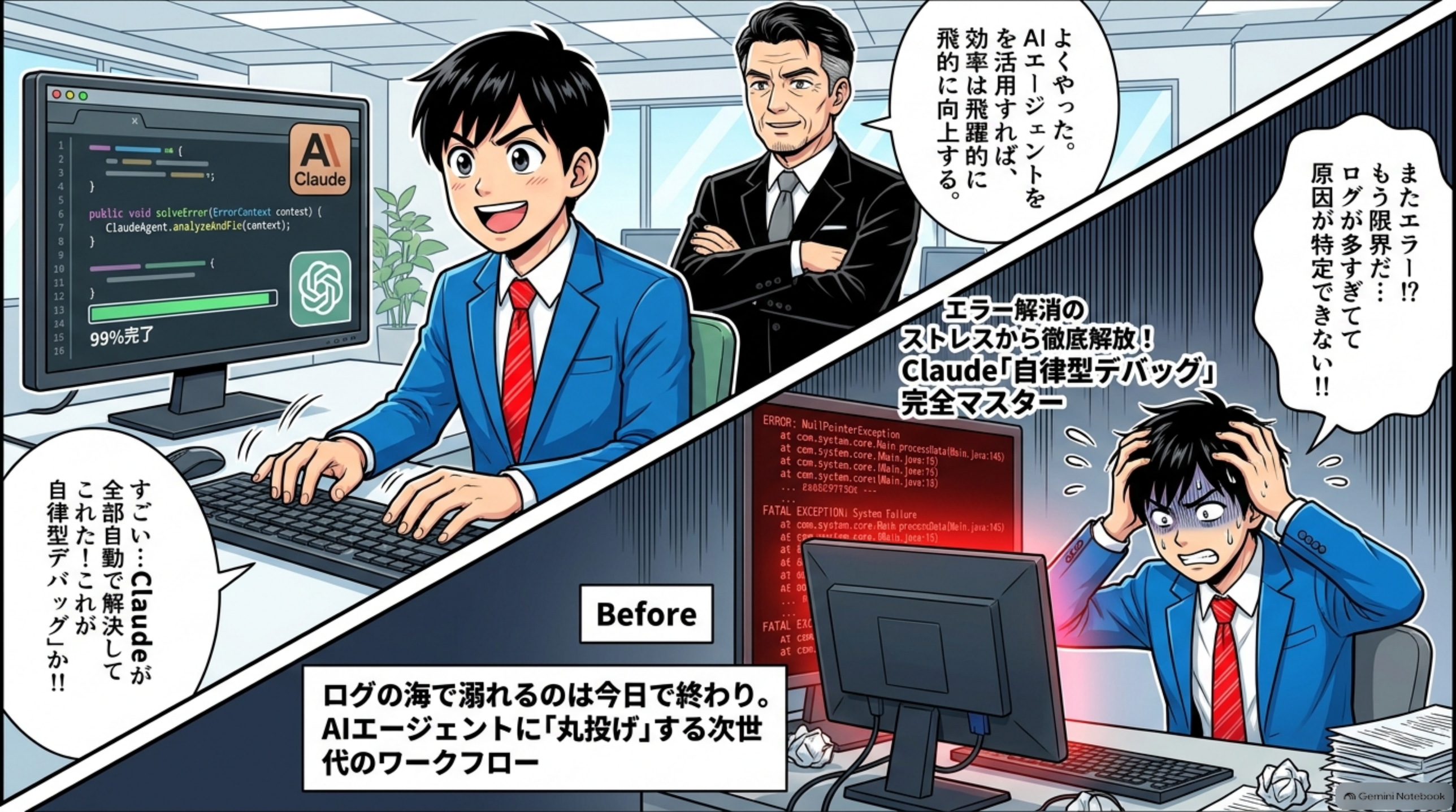
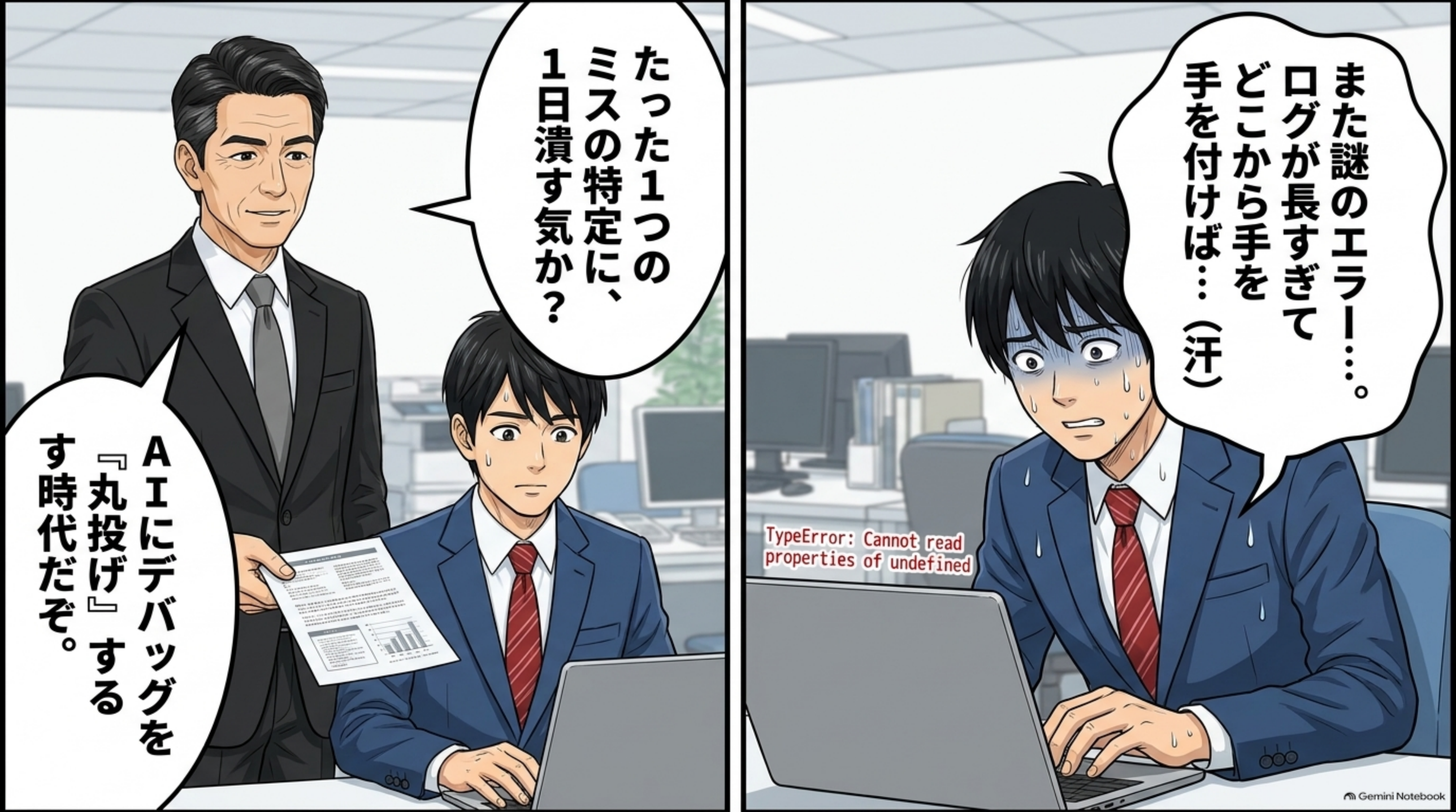
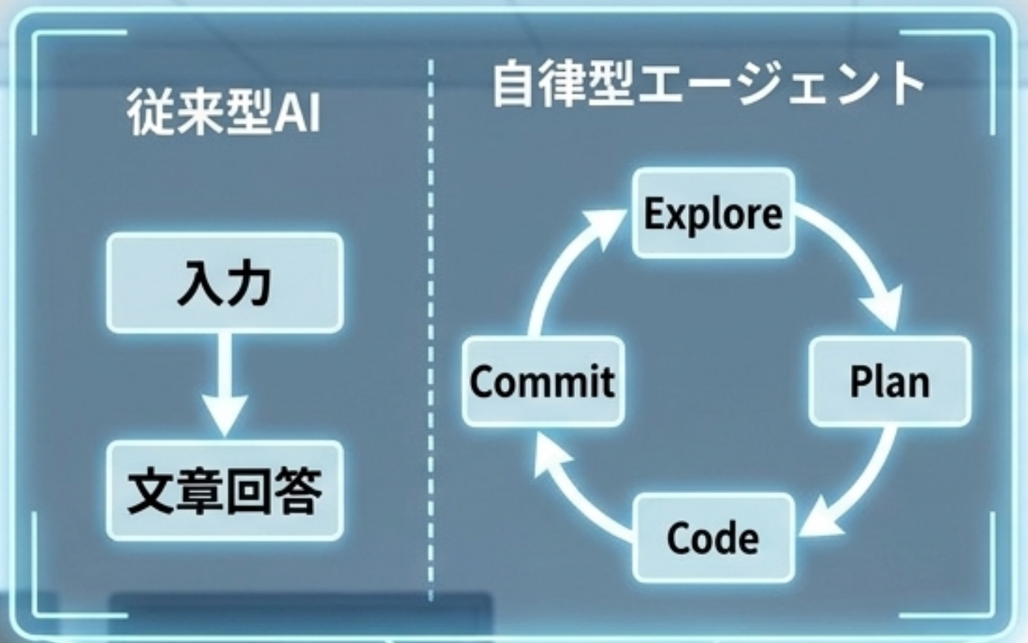

Claude
99%完了
よくやった。
AIエージェントを
を活用すれば、
効率は飛躍的に
飛的に向上する。
またエラー!?
もう限界だ…
ログが多すぎてて
原因が特定できない!!
エラー解消の
ストレスから徹底解放！
Claude「自律型デバッグ」
完全マスター
すごい…Claudeが
全部自動で解決して
これた！これが
自律型デバッグ」か!!
Before
ログの海で溺れるのは今日で終わり。
AIエージェントに「丸投げ」する次世
代のワークフロー
Gemini Notebook


また謎のエラー…。
ログが長すぎて
どこから手を
手を付けば…（汗）
TypeError: Cannot read
properties of undefined
たった1つの
ミスの特定に、
1日潰す気か？
AIにデバッグを
『丸投げ』する
す時代だぞ。
Gemini Notebook

でもAIって、質問に質問に答えてくれるだけですよね？大変だし…

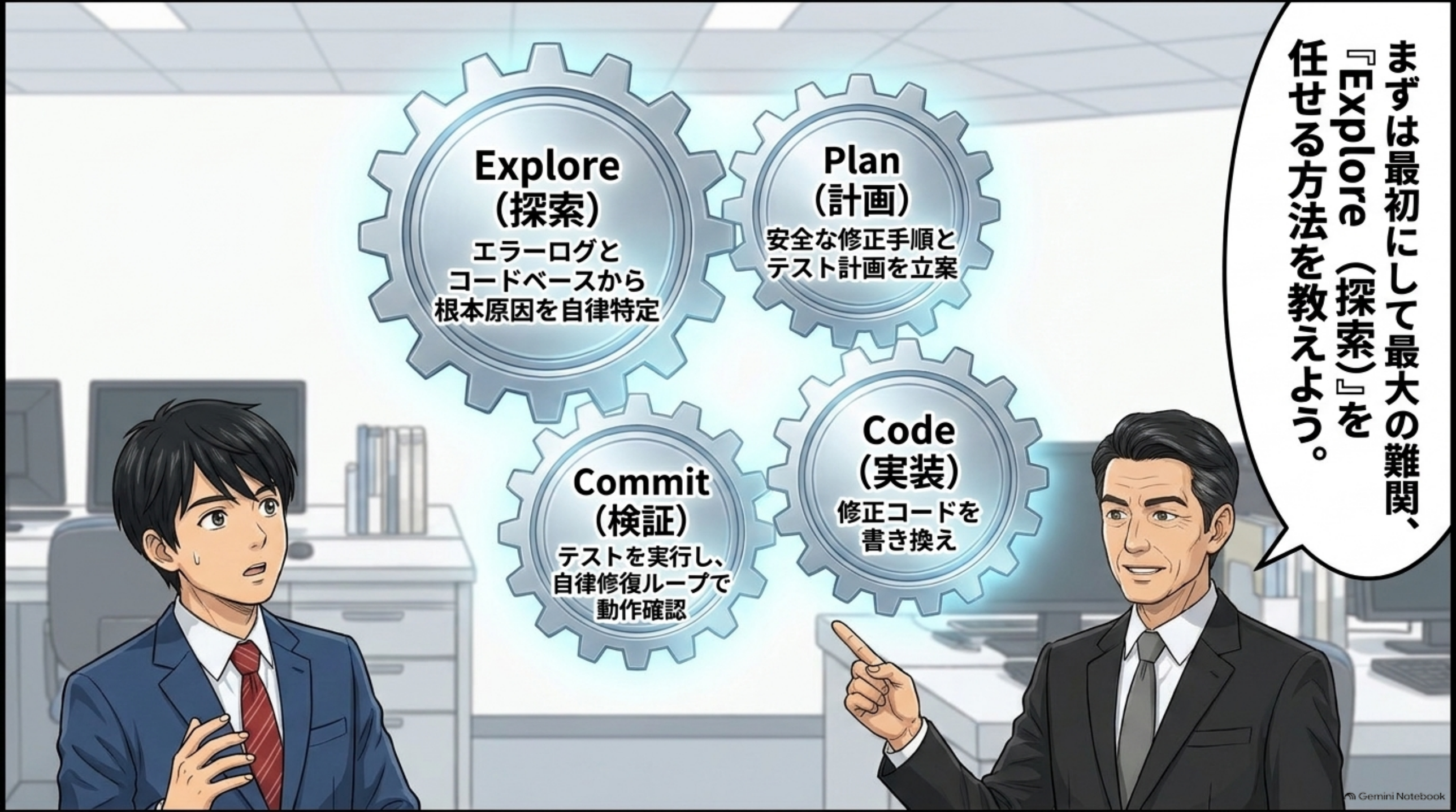
全部貼ドるのーをるコ大だし…

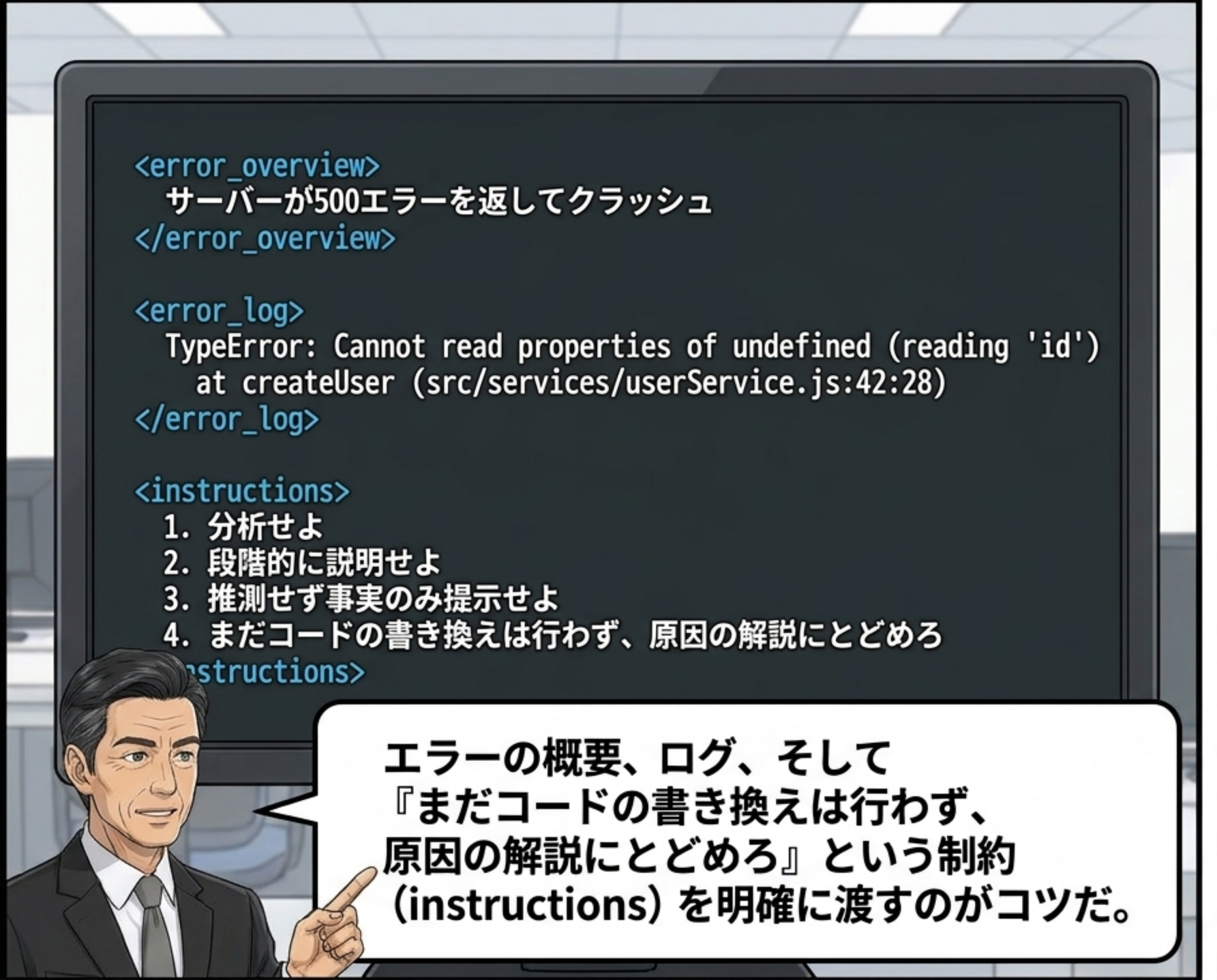
違う。Claudeは自律的にファイルを読み、推論し、テストまで実行す『エージェント』だ。



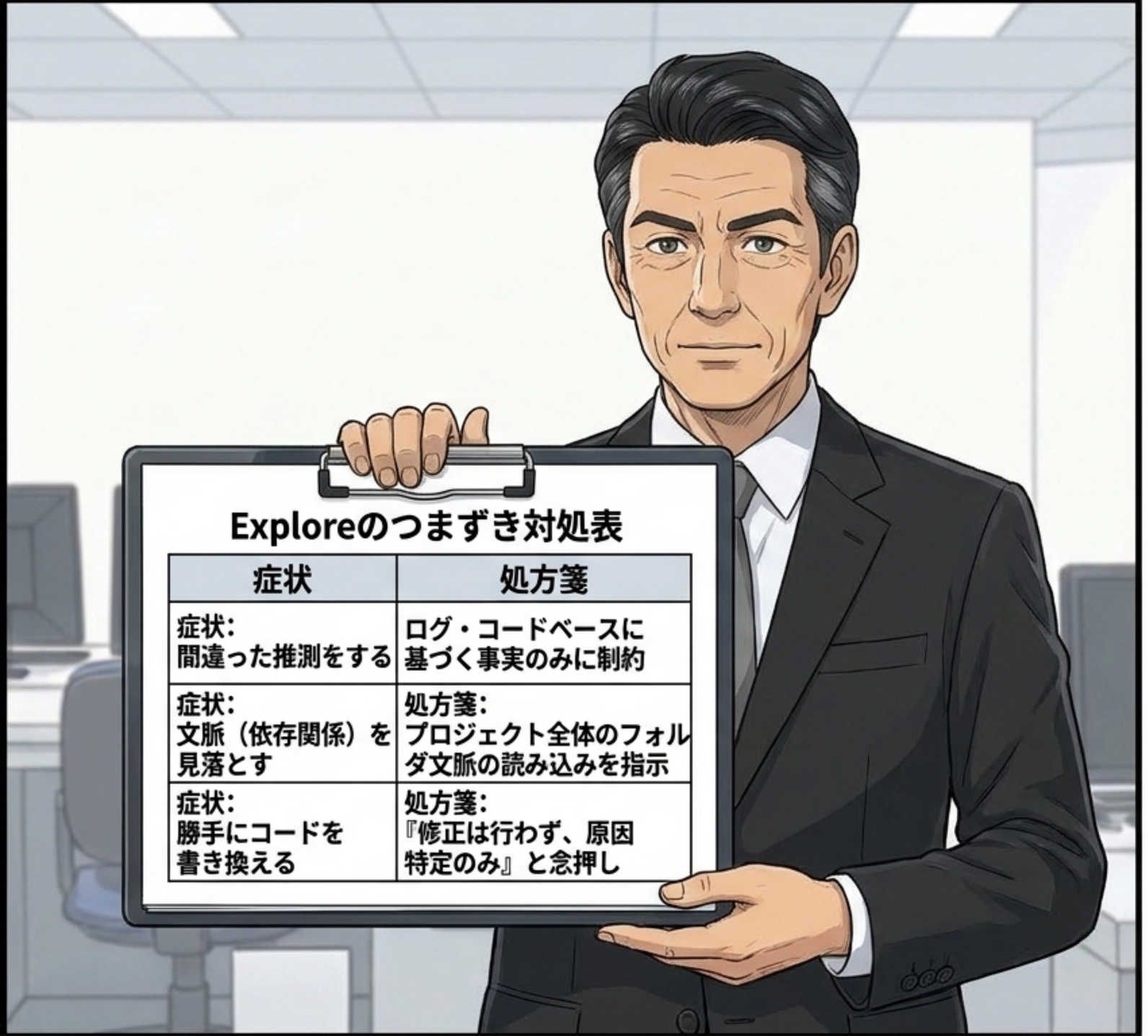
【Project Glasswingの実績】
OpenBSDで27年間潜伏していたクラッシュバグを
クラッシュバグを自律的に発見・修復。
プロのセキュリティ研究者と同等クラスの能力を持つ。

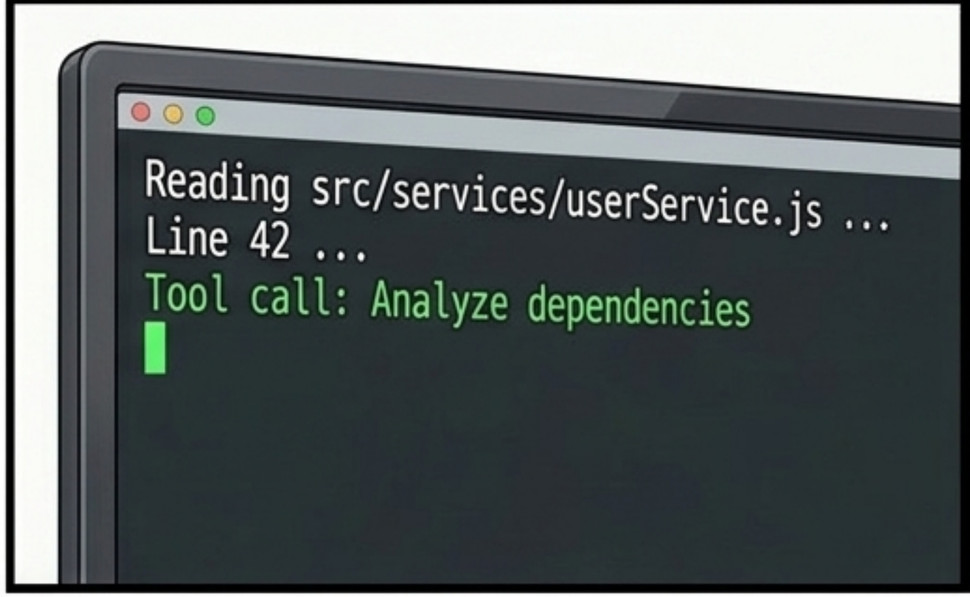
Generated with NotebookLM


まずは最初にして最大の難関、『Explore（探索）』を任せる方法を教えよう。
Explore（探索）
エラーログとコードベースから根本原因を自律特定
Plan（計画）
安全な修正手順とテスト計画を立案
Code（実装）
修正コードを書き換え
Commit（検証）
テストを実行し、自律修復ループで動作確認
Gemini Notebook


<error_overview>
サーバーが500エラーを返してクラッシュ
</error_overview>
<error_log>
TypeError: Cannot read properties of undefined (reading 'id')
at createUser (src/services/userService.js:42:28)
</error_log>
<instructions>
1. 分析せよ
2. 段階的に説明せよ
3. 推測せず事実のみ提示せよ
4. まだコードの書き換えは行わず、原因の解説にとどめろ
</instructions>
エラーの概要、ログ、そして
『まだコードの書き換えは行わず、
原因の解説にとどめろ』という制約
（instructions）を明確に渡すのがコツだ。

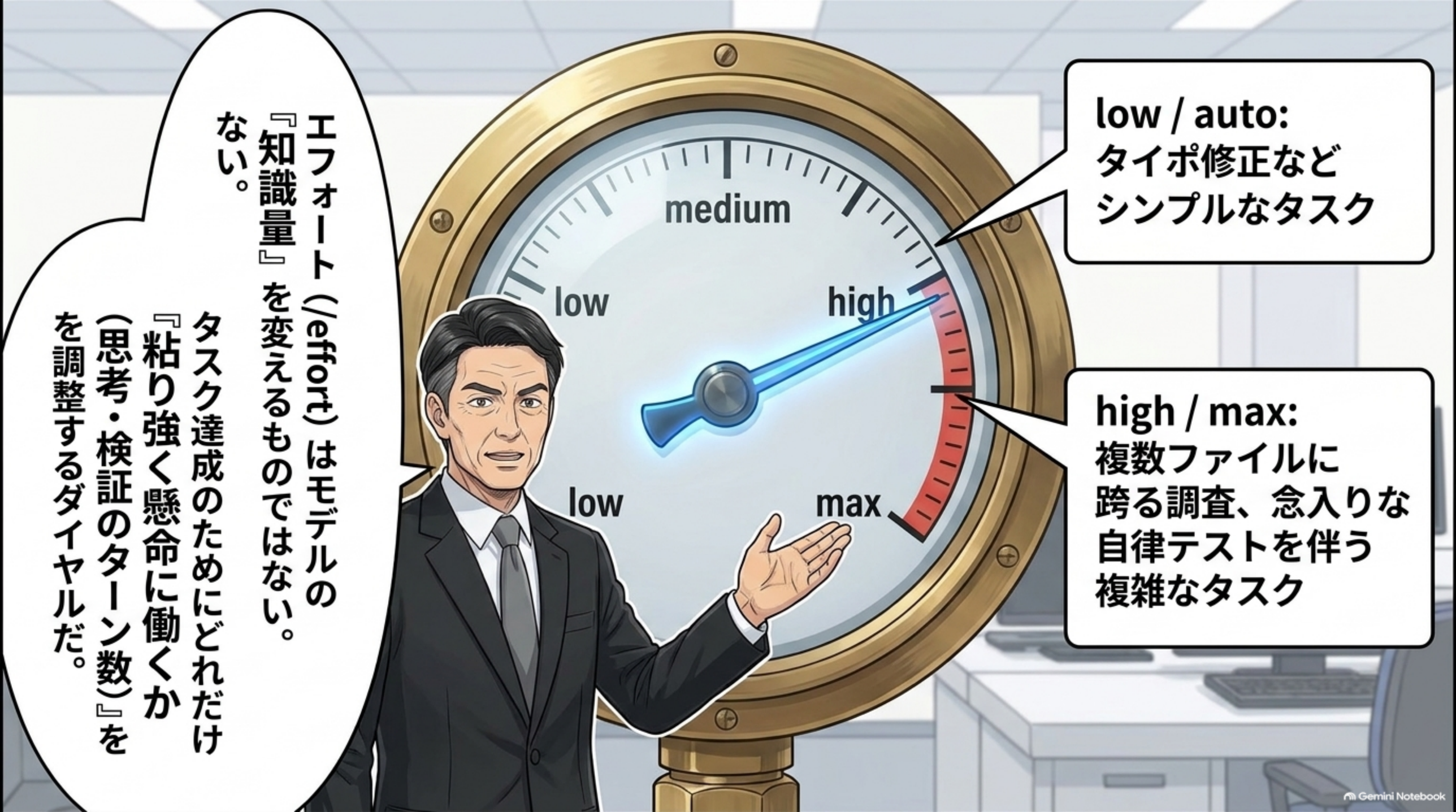
Gemini Notebook


Exploreのつまずき対処表
症状
処方箋
症状:
間違った推測をする
ログ・コードベースに
基づく事実のみに制約
症状:
文脈（依存関係）を
見落とす
処方箋:
プロジェクト全体のフォル
ダ文脈の読み込みを指示
症状:
勝手にコードを
書き換える
処方箋:
『修正は行わず、原因
特定のみ』と念押し


Reading src/services/userService.js ...
Line 42 ...
Tool call: Analyze dependencies


一瞬で根本原因
(undefinedになる理由)
を見抜いた…！
Gemini Notebook


よし、そのまま
修正（Code）させます！
待て！
いきなり修正させるな。
まずは『Plan（計画）』だ。
ガシッ
1. 変更するファイルの
ステップバイステップ計画
2. 既存機能への影響
（サイドエフェクト）の考慮
3. 修正後のテスト手順の明記
4. 承認するまで実際のファイル
修正は開始しないこと


low / auto:
タイポ修正など
シンプルなタスク
high / max:
複数ファイルに
跨る調査、念入りな
自律テストを伴う
複雑なタスク
medium
low
high
low
max
エフォート（/effort）はモデルの
『知識量』を変えるものではない。
タスク達成のためにどれだけ
『粘り強く懸命に働くか
（思考・検証のターン数）』を
を調整するダイヤルだ。
Gemini Notebook

効率的なAIの使い方

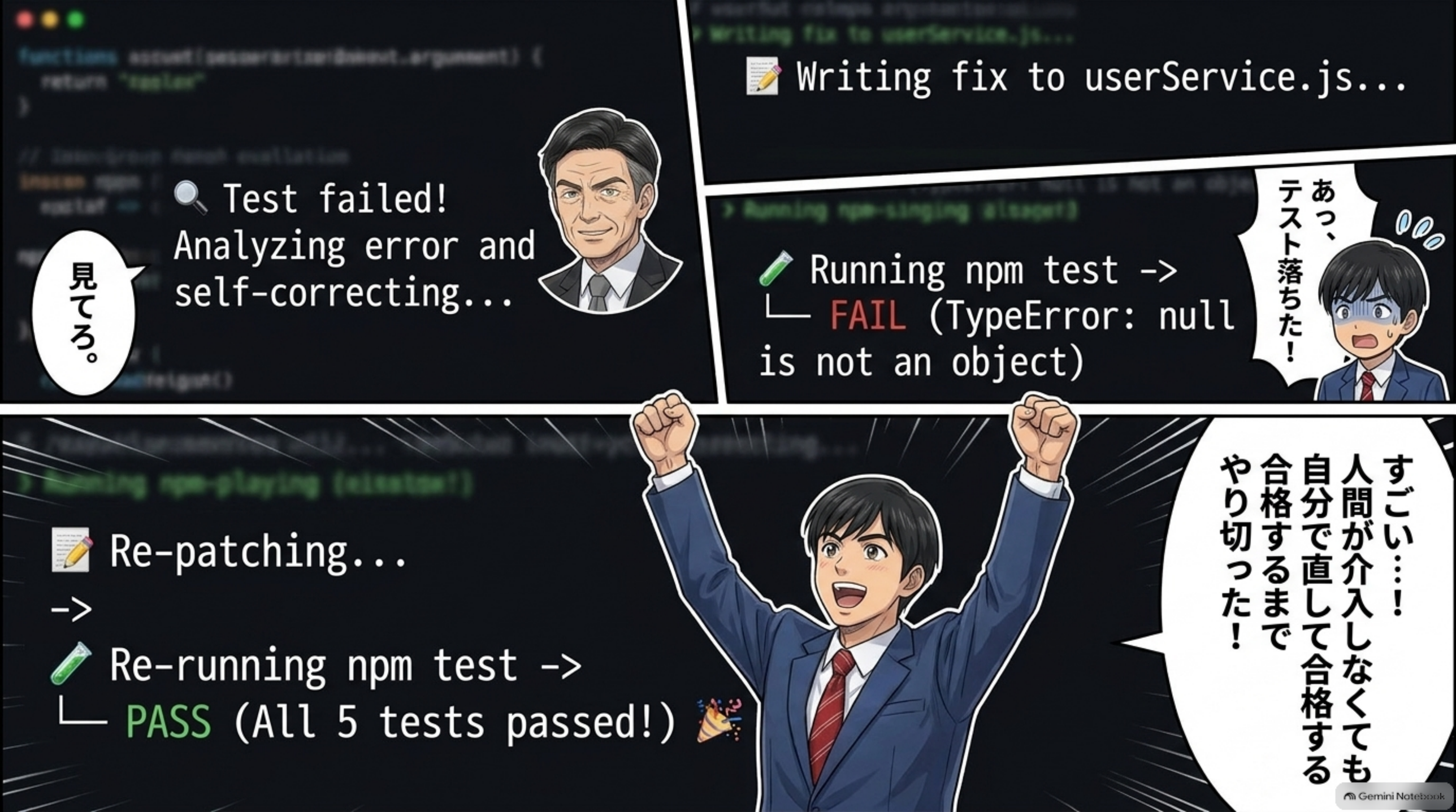


じゃあ、難しいバグは
『最高性能のモデル(/model)』にして
『エフォートmax』にすれば
にすれば最強ですか？

【超重要ルール】
より高性能なモデルに変更 (/model) した直後は、必ずエフォートを一度『デフォルト(/effort auto)』に戻せ。

解説：高性能モデルは推論能力が高いため、少ない思考トークン(低いエフォート) でも一発で解決できるケースが多い。

Gemini Notebook

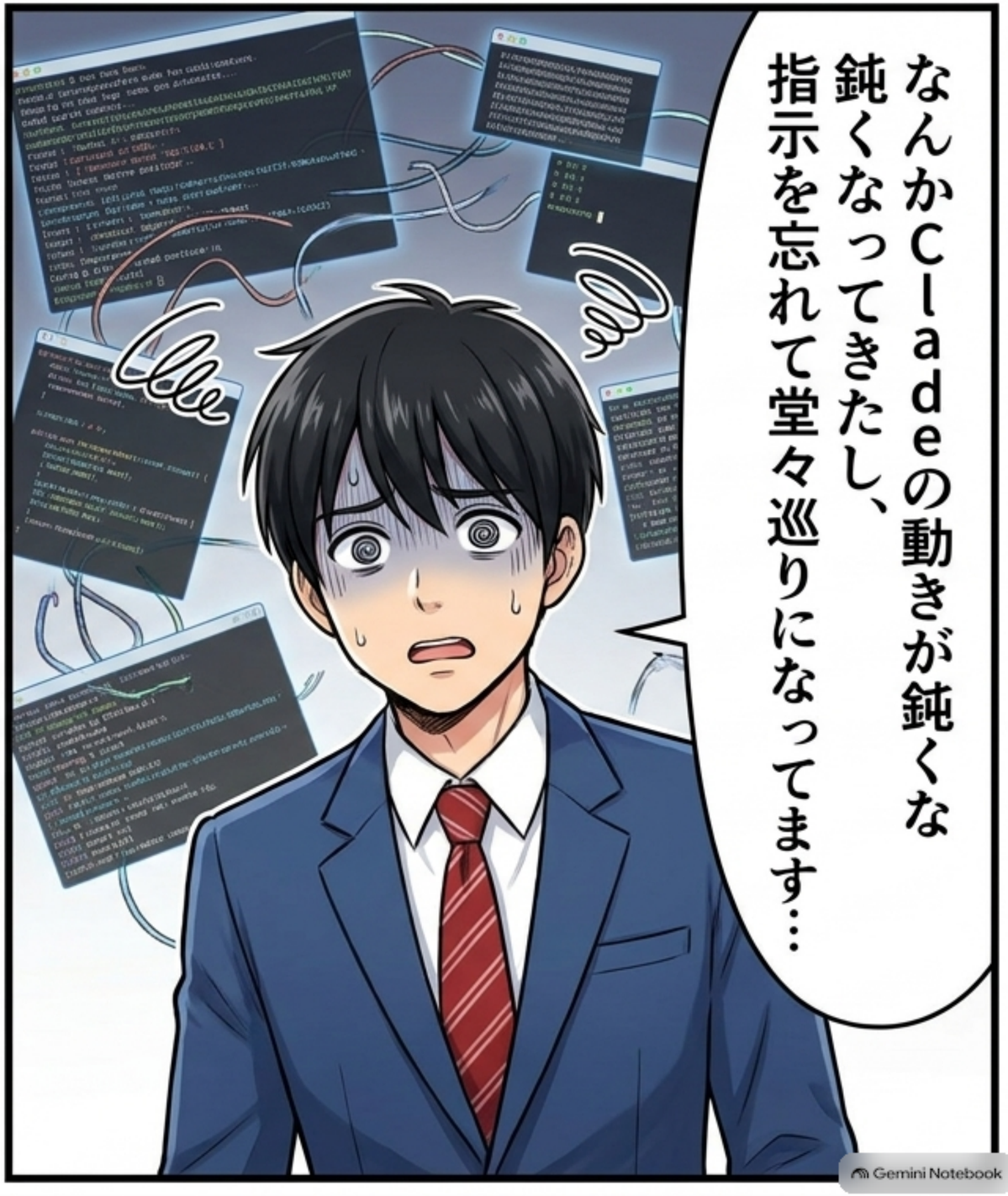

Writing fix to userService.js...
Test failed!
Analyzing error and self-correcting...
見てろ。
Running npm test ->
FAIL (TypeError: null is not an object)
あっ、テスト落ちた！
Re-patching...
->
Re-running npm test ->
PASS (All 5 tests passed!)
すごい…！
人間が介入しなくても
自分で直して合格する
合格するまで
やり切った！

デバッグ・トラブルシューティング表

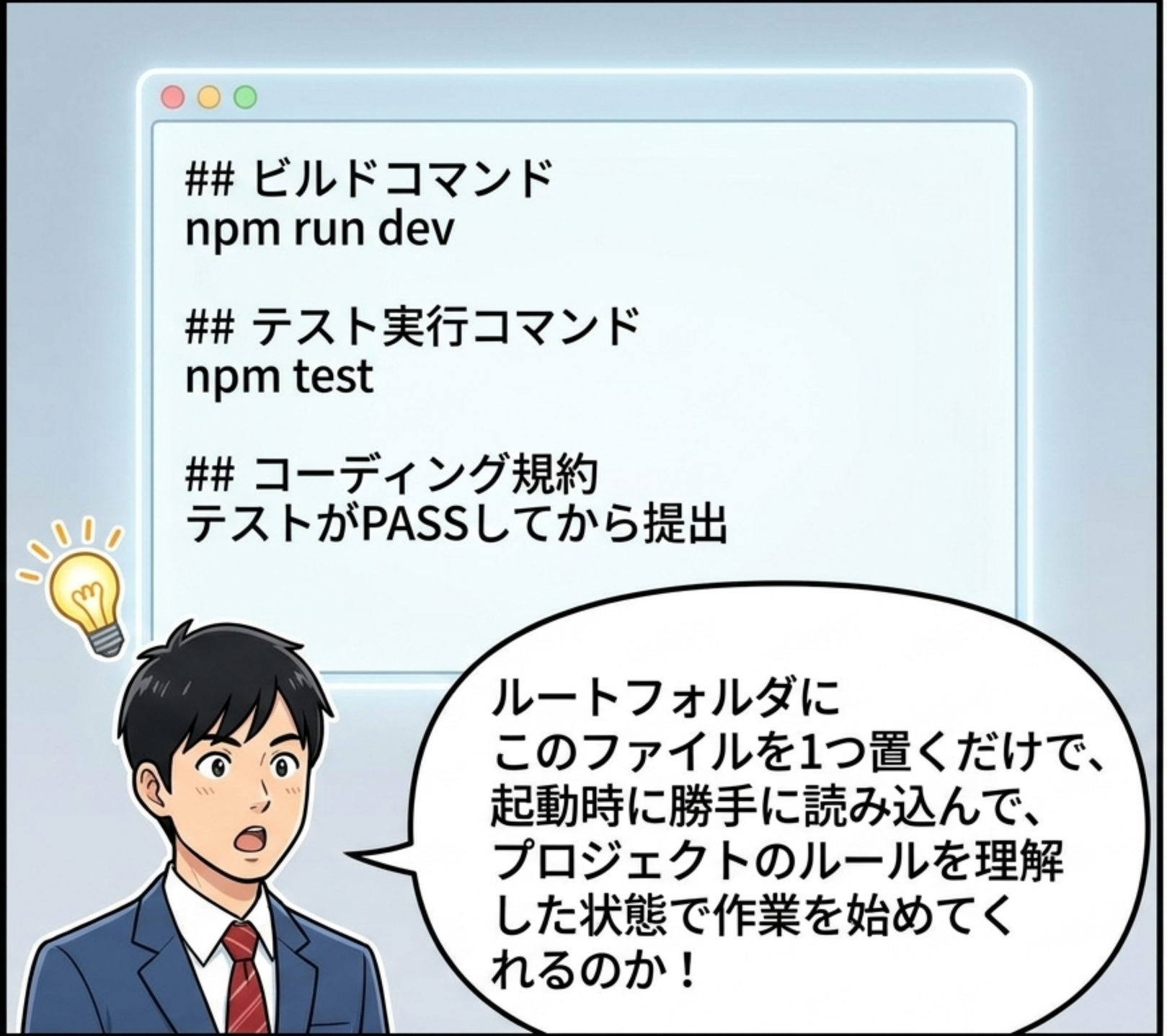
症状	原因	処方箋
途中で修正を諦めて未完成になる	エフォートレベルが低すぎる	/effort high または max に上げて再投入
考えすぎて時間がかかる・トークンを浪費する	タスクに対してエフォートが高すぎる	/effort low や auto に下げる
エフォートmaxでも正しく修正できない	モデル自体の基礎知識の限界	/model で高性能モデルに変更（直後にエフォートはautoへ）

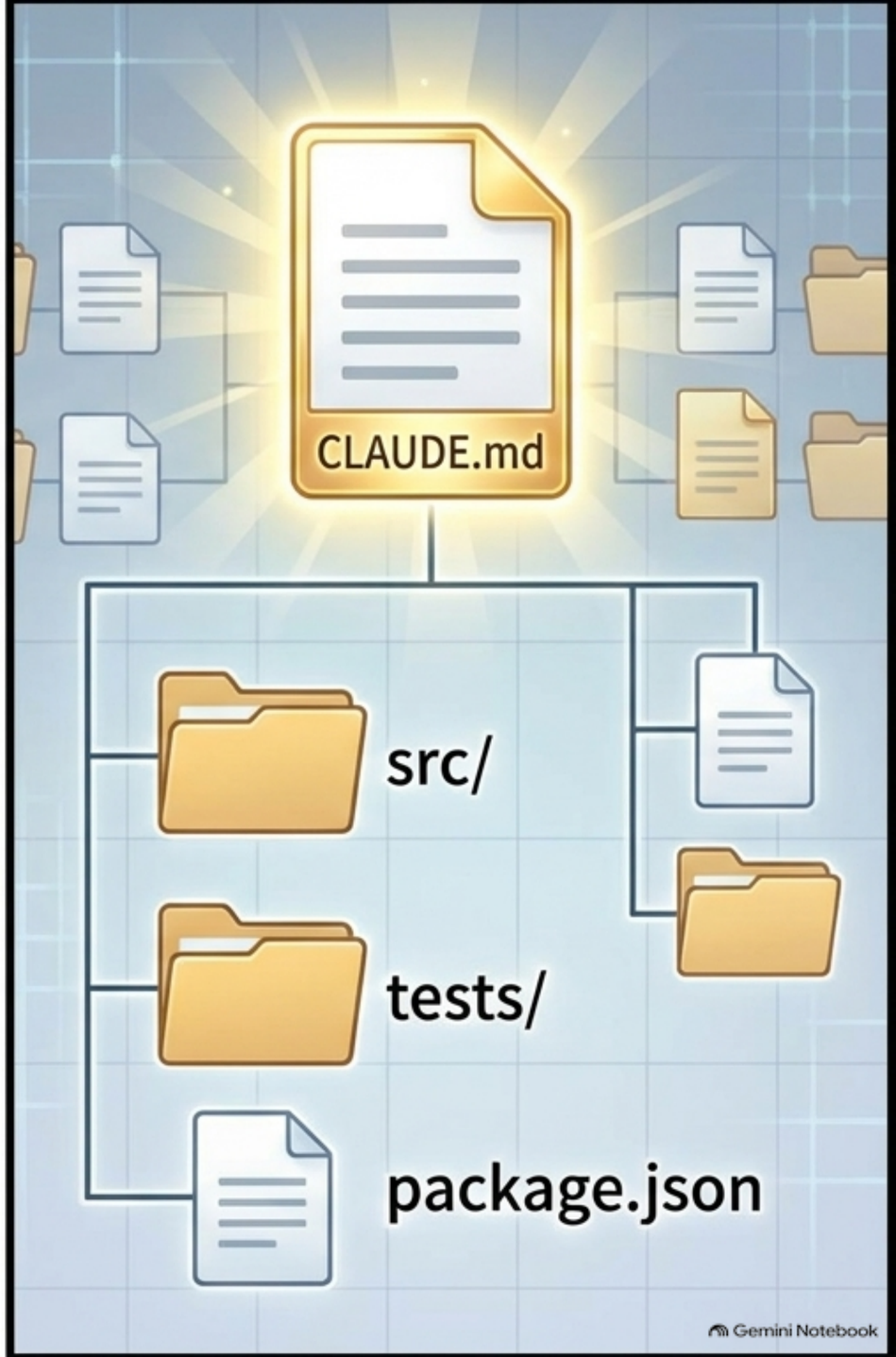
Gemini Notebook

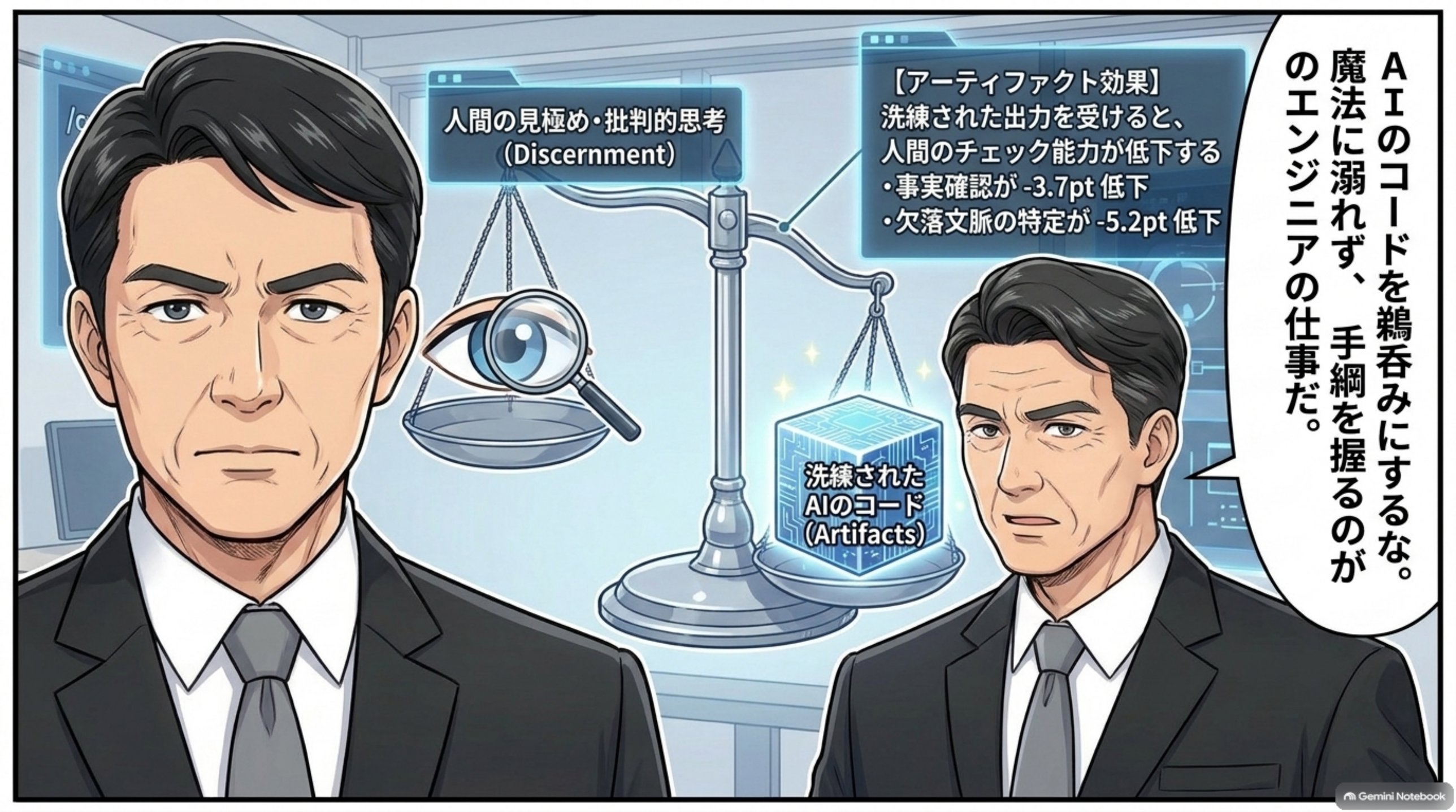




Gemini Notebook


ビルドコマンド
npm run dev
テスト実行コマンド
npm test
コーディング規約
テストがPASSしてから提出
ルートフォルダに
このファイルを1つ置くだけで、
起動時に勝手に読み込んで、
プロジェクトのルールを理解
した状態で作業を始めてく
れるのか！


CLAUDE.md
src/
tests/
package.json
Gemini Notebook


AIのコードを鵜呑みにするな。
魔法に溺れず、手綱を握るのが
のエンジニアの仕事だ。
【アーティファクト効果】
洗練された出力を受けると、
人間のチェック能力が低下する
・事実確認が -3.7pt 低下
・欠落文脈の特定が -5.2pt 低下
人間の見極め・批判的思考
(Discernment)
洗練された
AIのコード
(Artifacts)





Gemini Notebook