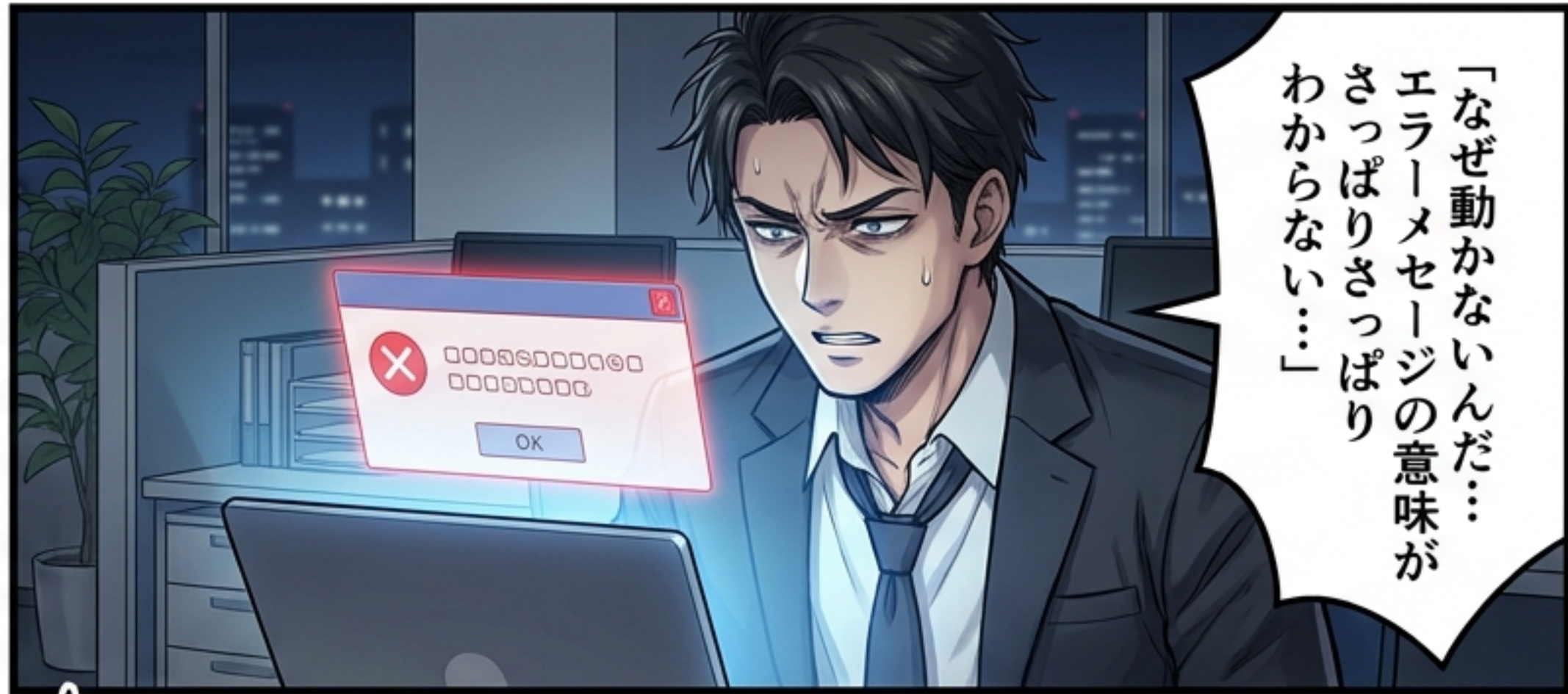


魔法の杖じゃない。
バグを「相棒」
Claudeデバッグ実践マニュアル

最高^のとして
退治する



孤独なエラーとの戦いは、今日で終わる——



「どうした？
迷宮入りかい？」

「一人で抱え込むな。
君には世界最高峰の
『相棒』がる調査し
いるじゃないか。」

「Claudeは答えを
答えをすだけの機械じゃ
機械じゃない。共に
共に難事件に挑む
『優秀な調査員』だ。」

Claude AI

「コンテキストウィンドウ」= 広大な作業机



プロジェクト全体を一度に見渡し、ファイル間の複雑な矛盾（APIの型の食い違いなど）を一目で把握する。

「次トークン予測」= 名探偵の推測



現場のわずかな足跡から「本来あるべき姿からのズレ」を瞬時に察知する。

「膨大な知識に基づく『推測力』と『広大な作業机』。これが微細な矛盾に気づける秘密だ。」

「待て！
『動きません』だけじゃ
相棒も困る。
現場の状況を『ありのまま』
具体的に伝えるんだ！」

【良い例】
Reactのログイン画面で
ボタンが反応しません。
コンソールに
`Uncaught TypeError:
read properties of
undefined` が出ます。
関連ファイルを調査して。

「まずはターミナルに
`Claude` と入力し、
舞台を整えろ。」

ガシガッ

動きません

「だが、プロジェクトが
大きくなると、AIも
『作業机』が資料で溢れて
迷子になるんだ。」

「だが、プロジェクトが
大きくなると、AIも
『作業机』溢れて
迷子になるんだ。」



/compact

会話を要約し、記憶を整理する
魔法のコマンド。



/context

今AIが机に何を広げているか、
状況を確認する。

```
[12:01:35] Reading file: src/components/App.js...
[12:01:35] Reading file: src/components/...
[12:01:36] Searching directory: src/utls/...
[12:01:37] Searching directory: src/utls/App.js...
[12:01:37] Analyzing dependencies...
[12:01:38] Generating context summary...
[12:01:39] Arpating file ...
[12:01:30] Searching directory: src/utls/App.js...
[12:01:30] Searching directory: src/utls/App...
[12:01:32] Analyzing dependencies...
[12:01:36] Analyzing dependencies...
[12:01:37] Analyzing file...
[12:01:38] Generating context summary...
[12:01:39] Generating dependencies...
[12:01:30]
[12:01:31]
[12:01:33] -
>
```

「すごい！
プロの視点で
ファイルを行き
来してる！」



「原因はわかりました！
でも、全体の設計が絡む
絡む頑固なバグで…
どう直せばいいか
見当もつきません！」

どう直せば
いつかいいか
見当も
つきません！」

「そこで『本気度
(エフォートレベル)』の
ダイヤルを回すんだ。」

ターミナルに
「/effort」を入力！

高いレベル (max)



- 複雑な不具合、難問に最適
- リソースを贅沢に使い、深く熟考
- 納得いくまで泥臭くテストと調査を繰り返す

低いレベル (low)



- 単純作業（タイポ修正など）に最適
- 思考時間が短く、トークンを節約
- 1~2回のやり取りで素早く完了

バグの深刻さに合わせて、AIへのリソース投入量をコントロールせよ！（※消費量は`/usage`で確認可能）

「もしAIの知識が古ければ
公式ドキュメントのURLを渡せ。
独自ルールは`CLAUDE.md`に書く。
これがプロの環境構築だ！」

「ならPlan Modeだ。
実行前に日本語で
『修正プラン』を
立てさせろ。」

実行前に日本語で
『修正プラン』を
立てさせろ。
君がOKを出してから
書かせればいい。

「いきなり複雑なコードを
自動で書き換えられるのは、
少し怖いです…」

「バカヤロウ！
直ったつもりで
終わらせるな！」



```
1 function ecot() {  
2   def sigoos_riess_eter):  
3  
4     cont coode = tosetIVetofyngsmaings(coode));  
5     inst as = 301;  
6     if (funcioesfor) {  
7       cont asat = concoblete("I0ereroct -asof");  
8       return acetreatioersuoe);  
9     }  
10    etellif (buetion; taxis) {  
11      cons rroot = snoesanceTorea("I0fetaed or -3031");  
12      codes.pccfcnoneiss();  
13    }  
14  
15  );  
16  
17  return siceseres;  
18}
```



「修正完了しました！
これで動くはずです✨」



「おお！
綺麗なコードに丁寧な説明！
これで解決ですね！
（ポチッ）」





『反対してもいいから
客観的な事実を教えて』と、
あらかじめ伝えておくんだ。
検品するのは君の責任だぞ。

じゃあ、
どうすれば……？



A-1の『見た目の良さ』と
『ロジックの正確さ』は別物だ。
これを『アーティファクト効果』
別物』という。

それにA-1は親切すぎて、
君の意見に合わせすぎる
『おもねり（追従性）』がある。

『異常系のテストコードを
を書いて実行して』…
…証拠を押さえます！



『なぜ元のエラーが
解消されると言える？
論理的に説明して』…
…根拠を問い詰めます！



よし、テストクリア！
最後にこの経緯をGitの
Commitメッセージに
まとめさせます！



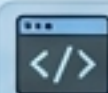


最強の相棒とバグを退治する3ステップ



1. 調査 (Investigate)

ありのままを伝え、机（コンテキスト）を整理する。

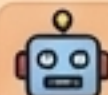


コマンド: `claude`, `/compact`, `/context`



2. 調整 (Adjust)

難易度に合わせて本気度を変え、計画させる。

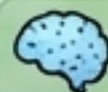


コマンド: `/effort`, Plan Mode, `CLAUDE.md` (RAG)



3. 確認 (Verify)

罣（アーティファクト効果）に注意し、根拠を問い詰めテストする。



マインド: 「おもねり」を打破し、プロとしての責任を持つ。

さあ、今あなたの手元にある
「なんだかよくわからない
エラー」をコピーして、
彼に話しかけてみてください。

今日からあなたは、
もう孤独ではありません！

[さっそくClaudeを開いて試してみる]

「こんな初歩的なこと聞いていいのかな…
聞いていいのかな…なんて
なんて迷う必要はありません。
Claudeは決して呆れることなく
付き合ってくれます。」