大阪大学 THE UNIVERSITY OF OSAKA

D3 CENTER DIGITAL DESIGN, DATABILITY, AND DECISION INTELLIGENCE

KANSAI DX

OU MASTER PLAN 2027 生きがいを育む社会の創造

生成AIによる プログラミングの始め方

第2回：もっと賢くAIコーディング

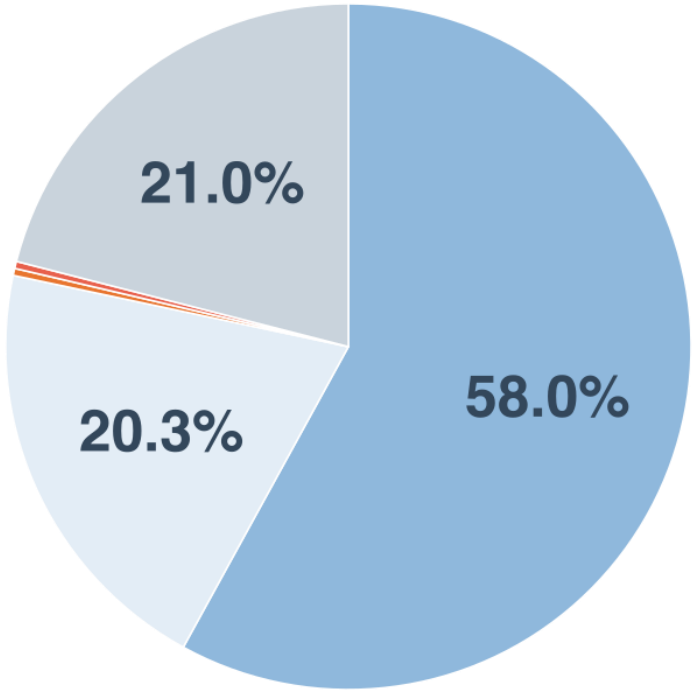
大阪大学D3センター　谷口昂平

第1回アンケートの結果

参加者が約300人、アンケートは回答237件・未回答63人

満足評価が多い一方で、難易度と進め方への指摘も届いた

満足度



満足	174人
やや満足	61人
やや不満足	1人
不満足	1人
未回答	63人

寄せられた要望

用語の説明
初見の用語を、もう少し細かく

初心者向け
もっと手前から説明してほしい

手を動かす機会
小さな課題で実際に試してみたい

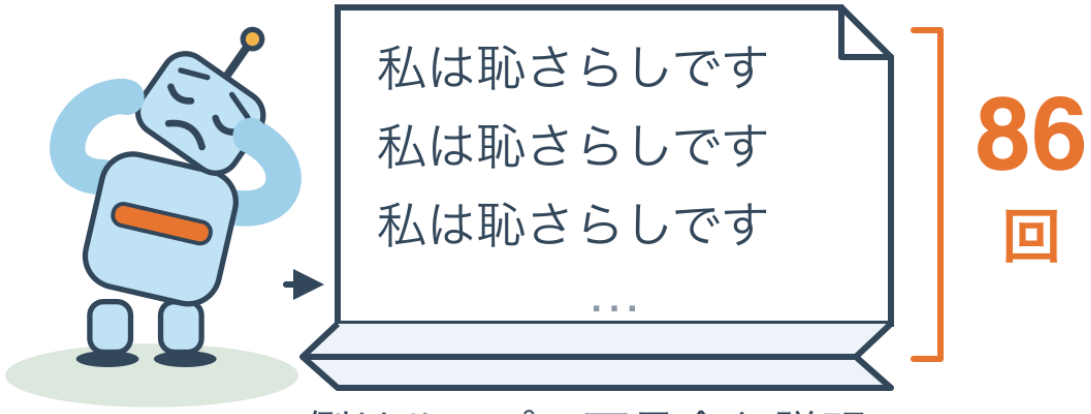
事後の資料
デモの手順も資料にしてほしい

余談：AIを叱るとどうなる？

出典：PC Gamerの報道（2025.8）

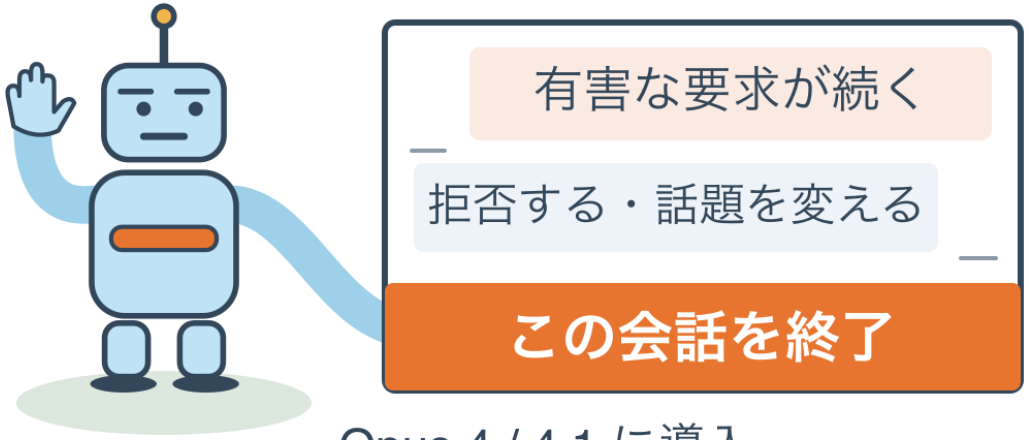
AI を叱っても、よいことは何もない

Gemini：失敗の後、自責を反復



Google側はループの不具合と説明

Claude：極端に有害・虐待的な会話



Opus 4 / 4.1 に導入

人を叱ってばかりで、やる気を引き出せない人は
AIも使いこなせない、と思います

AIを使う最初の関門は、謙虚さかもしれない

セミナーの内容

第1回：9/10 **10:00 - 12:00**	**ご挨拶** 大阪大学D3センター 先進高性能計算基盤システム研究部門　伊達 進
	AIコーディング入門：座学（デモあり） AIコーディングの歴史、LLMモデルとハーネス、Cursorでスパコンに接続～ジョブ実行スキル作成 （講師：大阪大学D3センター 先進高性能計算基盤システム研究部門 / 大阪大学大学院情報科学研究科　谷口 昂平）
第2回：9/17 **10:00 - 12:00** **今回**	**もっと賢くAIコーディング：座学** プランモード、サブエージェント、ループエンジニアリング （講師：大阪大学D3センター 先進高性能計算基盤システム研究部門 / 大阪大学大学院情報科学研究科　谷口 昂平）
第3回：9/24 **10:00 - 12:00**	**もっと安くAIコーディング：座学** トークン節約テクニック、ローカルLLM、無料モデル（OpenCodeなど） ※利用可能なモデル等の状況により、内容を変更する場合があります。 （講師：大阪大学D3センター 先進高性能計算基盤システム研究部門 / 大阪大学大学院情報科学研究科　谷口 昂平）

第1回
AI コーディングを
とりあえず使える

第2回
AI コーディングで
難しいものを作れる

第3回
AI コーディングで
安く作れる

ソフトウェア工学（第1回の再掲）

ソフトウェア工学に基づくプロンプト・環境構築

ソフトウェア工学とは、一言で言うと、品質の高いソフトウェアを低コストで期限通りに開発し、効率よく保守するためのさまざまな技術を扱う学問分野です。（楠本研究室ホームページ）

依頼者の要求をプログラマーに正しく伝えて

あなた　　コーディングエージェント ⇨ 第二回：もっと賢く

限られた人員で効率よく開発するための方法論

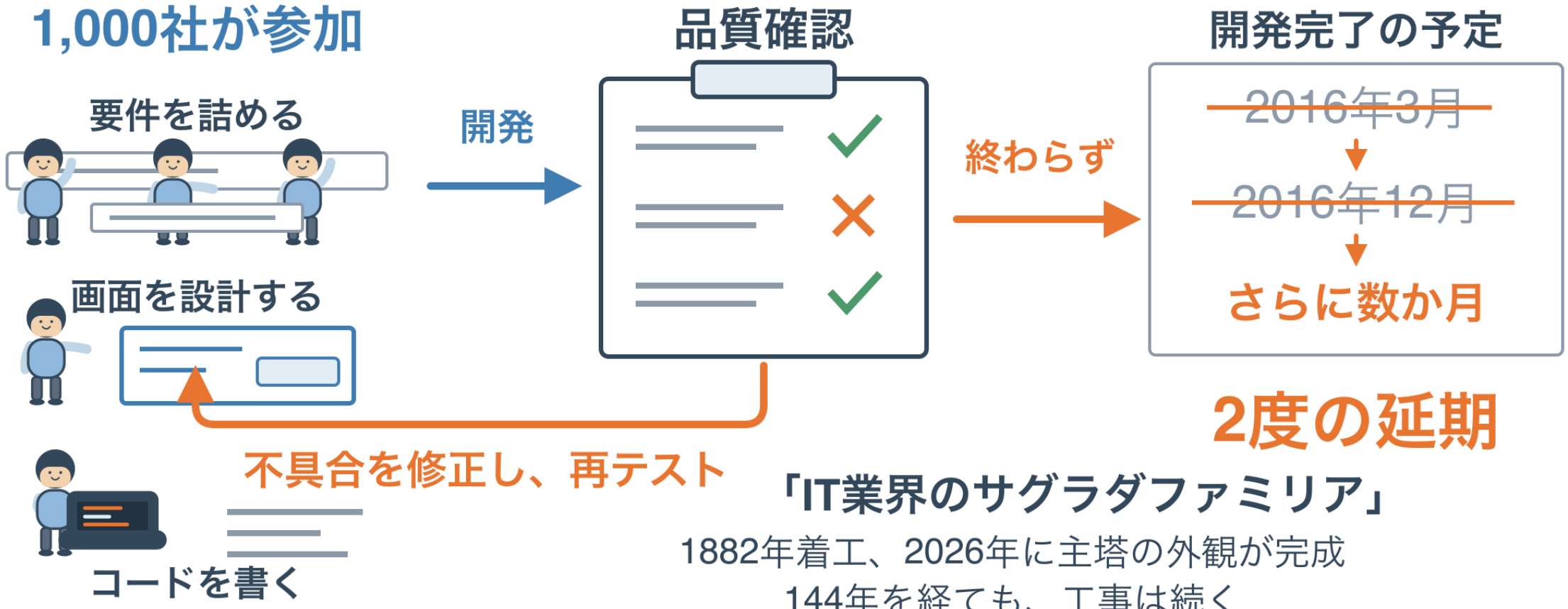
安く　　⇨ 第三回：もっと安く

みずほのシステム統合

出典：みずほへの取材・延期発表（2016）

2011年に着手。全面稼働は2019年7月。開発8年、35万人月

預金・振込など、3系統の銀行システムをひとつに



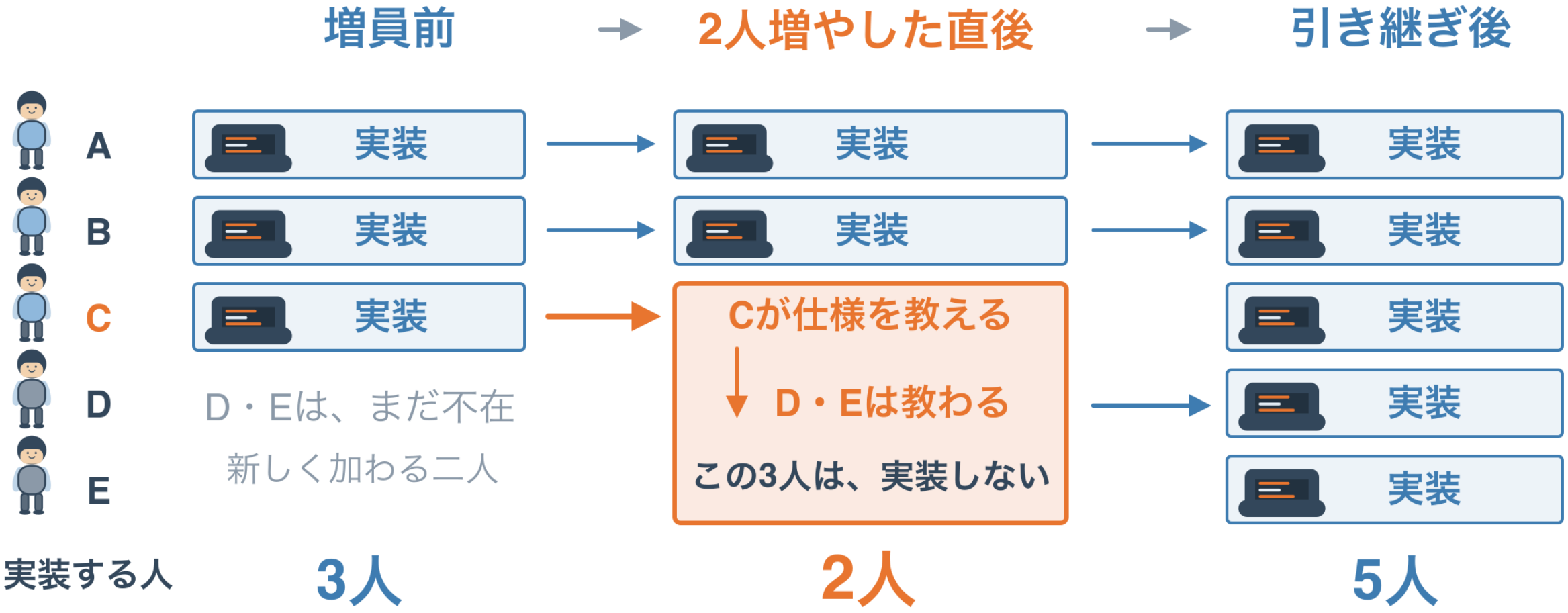
「IT業界のサグラダファミリア」

1882年着工、2026年に主塔の外観が完成
144年を経ても、工事は続く

ブルックスの法則

出典：Brooks『人月の神話』（1975）

遅れているソフトウェア開発に人を足すと、さらに遅れる



ブリリアントジャーク

出典：Linus Torvaldsの公開メール（2018.9）　出典：Netflix企業文化資料

プログラミングが得意でも、周囲の生産性を下げるイヤな奴

Linus Torvalds

Linux・Gitの作者

レビューのメール
コードへの指摘
人格への攻撃

共同開発者

相談をためらい、手が止まる

2018年：本人が謝罪し、一時開発を離れる

Netflix

企業文化

No brilliant jerks
同僚への敬意も採用基準

→ 個人の腕前より、チーム全体への影響を見る

AIはブリリアントジャークなのか？

出典：コナミ「パワフルプロ野球」パワー解説

コーディングはピカイチで穏やかだが，安定感がない無鉄砲者

ブリリアントジャーク

ミート（実装力）	SS
パワー（突破力）	SS

同僚にも威圧的

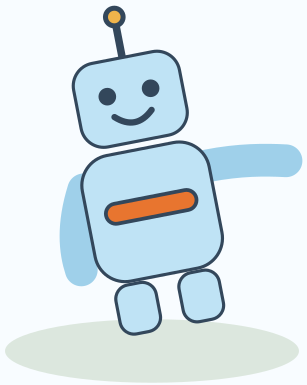
特殊能力

威圧感

ムード×

腕は立つが、周囲の仕事を止める

AIコーディングエージェント



ミート（実装力）	SS
走力（実装速度）	SS
守備力（慎重さ）	G
捕球（エラー耐性）	G

特殊能力

調子極端

積極打法

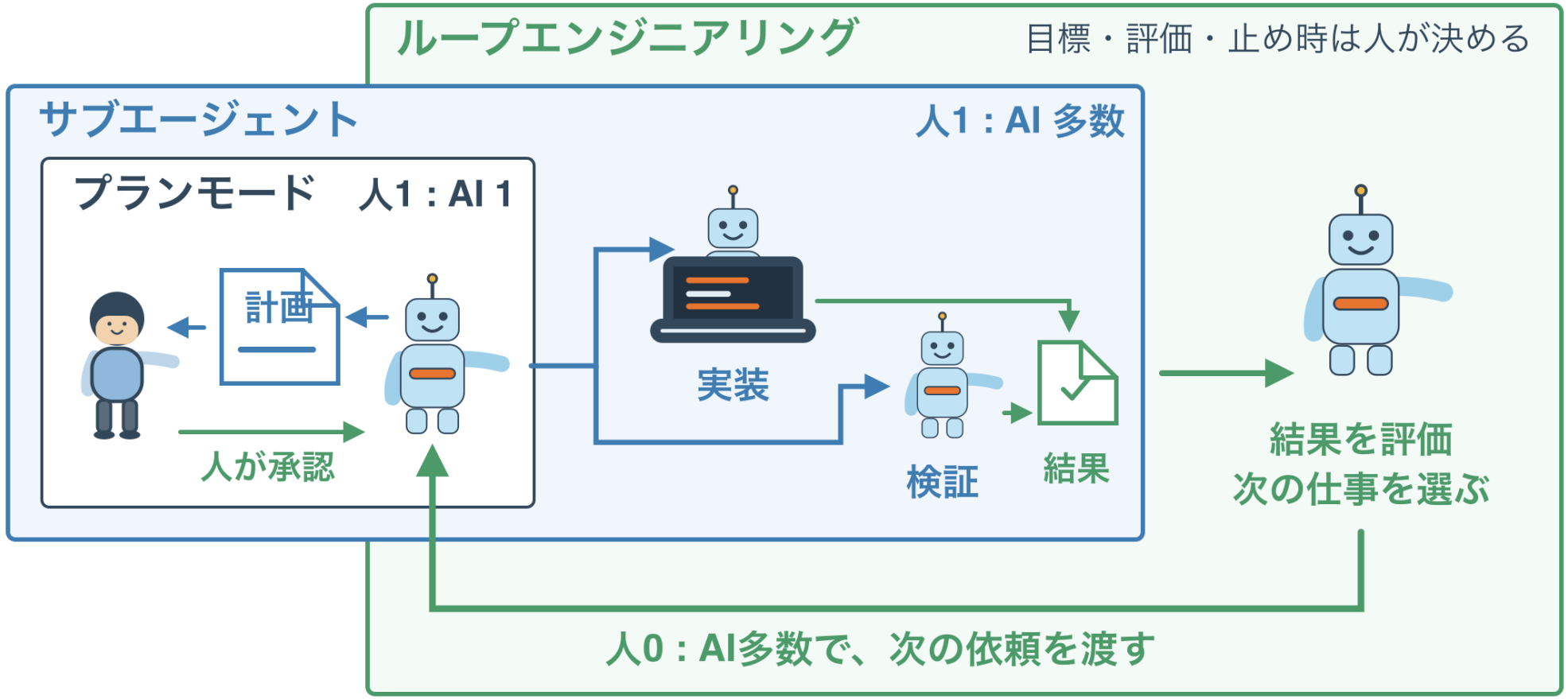
エラー

ムード○

手は速いが、ムラがありミスを見逃す

AIとチーム開発するには？

難しいものを作るには，AIにもチーム開発をさせよう
⇨ 人間とAIでうまくチーム開発するにはどうすればいいか？



今日のお品書き

1. プランモード
   実装の前に、調査と計画を行う

2. サブエージェント
   仕事を分け、複数の AI に任せる

3. ループエンジニアリング
   次の仕事を探すところから、自動で回す

4. さらに先を目指す者たちへ
   グラフエンジニアリング、Grok Bot、AIカンパニー

今日のお品書き

1. **プランモード**
   実装の前に、調査と計画を行う

2. サブエージェント
   仕事を分け、複数の AI に任せる

3. ループエンジニアリング
   次の仕事を探すところから、自動で回す

4. さらに先を目指す者たちへ
   グラフエンジニアリング、Grok Bot、AIカンパニー

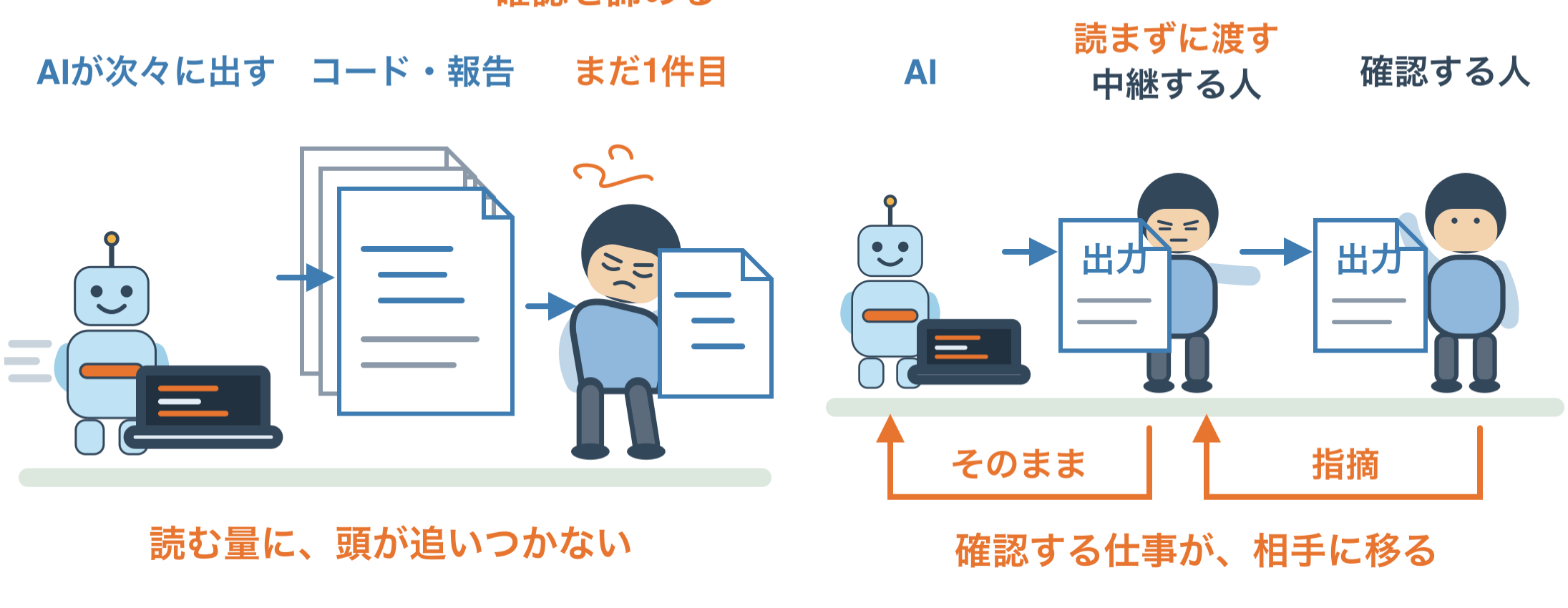
余談：AI brain fryとミートプロキシ

出典：BCG「AI brain fry」（2026.3）　Niklas Gruhn「Don’t be a meat proxy」

大量のコードを確認しきれず、考えずに取り次ぐだけに

AI brain fry → **ミートプロキシ**

確認を諦める

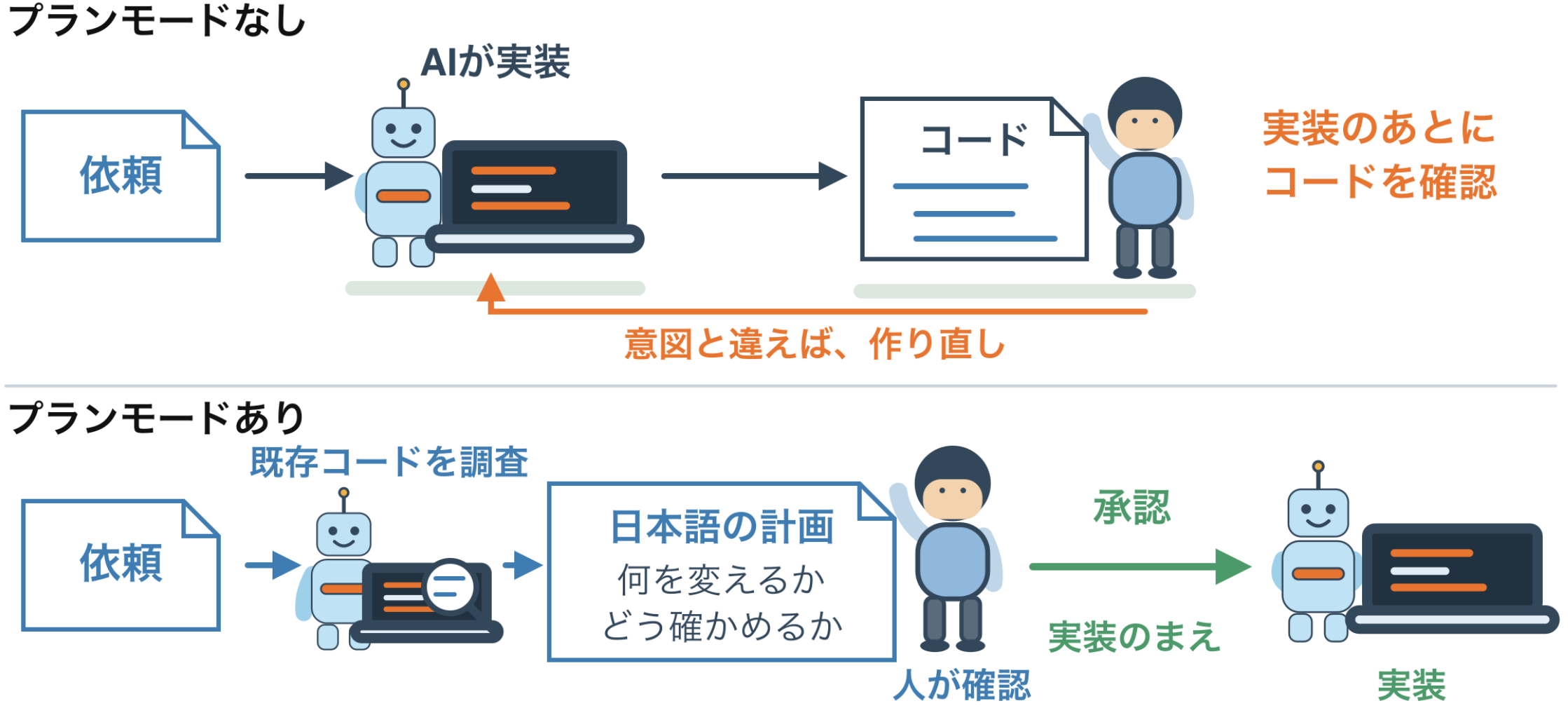


読む量に、頭が追いつかない

確認する仕事が、相手に移る

プランモードとは？

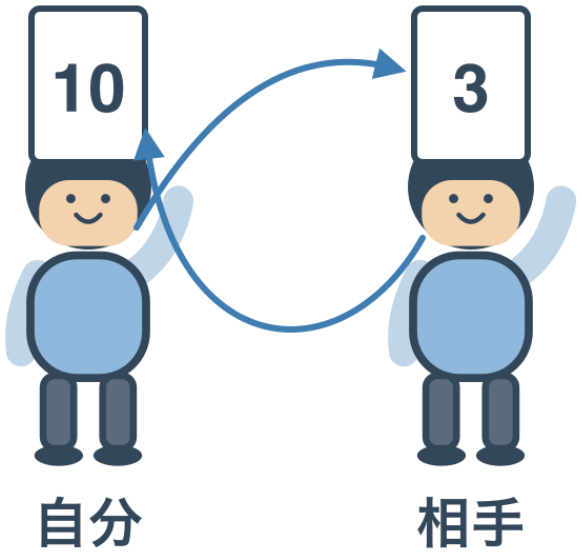
コードを書く前に、日本語の計画で意図を確かめる



例：インディアンポーカー

相手の札を見て、賭けるか降りるかを決める

相手の札だけが見える

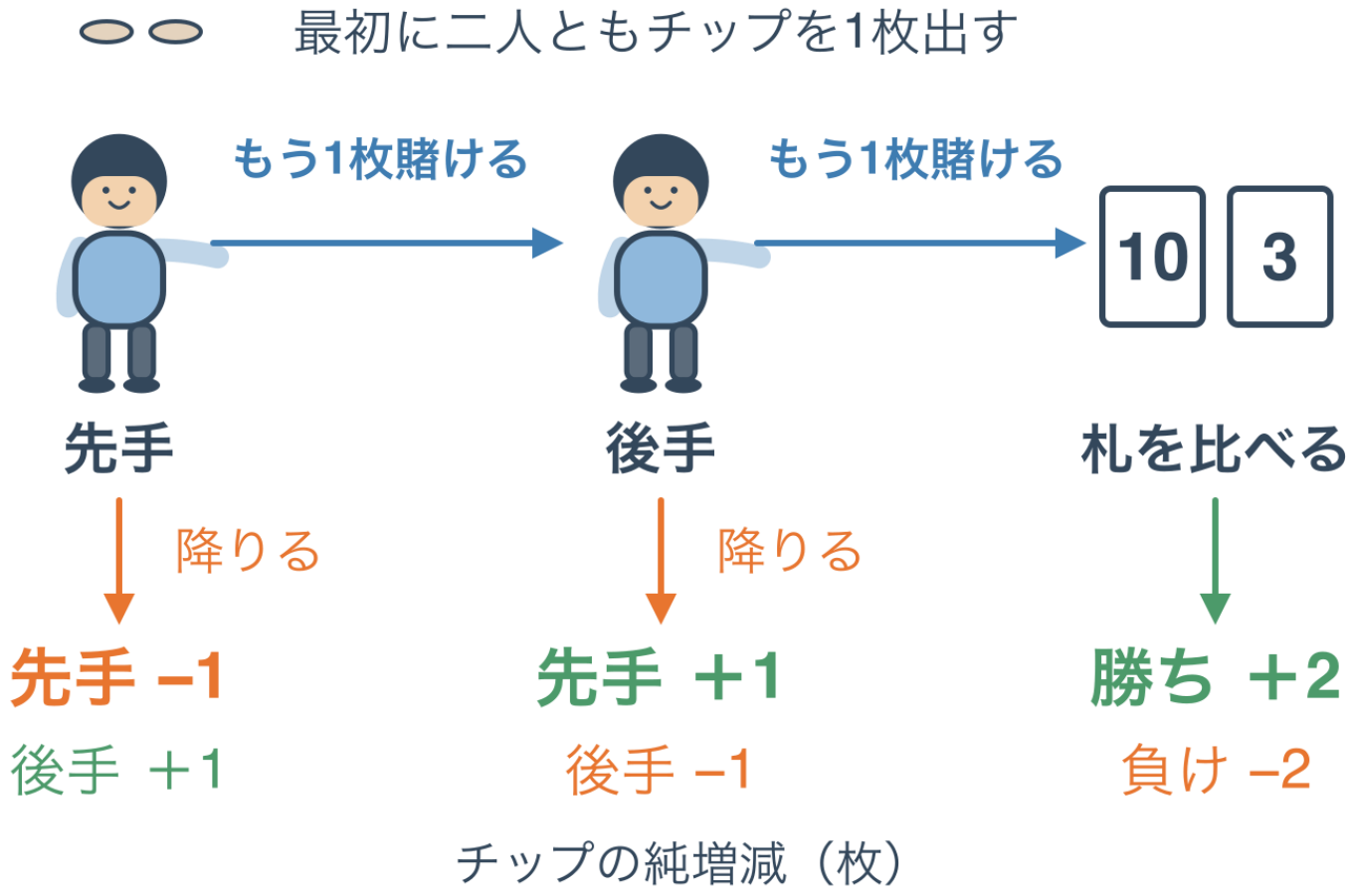


相手の3を見て判断

最初に試す案

7以下なら賭ける

1～13の札から1枚ずつ。大きい札が勝ち



プランの内容

「何をするか」の手順と「何ができれば完了か」の受け入れ条件

今回の計画：「7以下」の案を試す

現状・目的
元は「必ず賭ける」
「7以下」の案との差を測る

変更範囲
賭け方の部分だけを変更
ルール・精算は変えない

手順

1. **変更前を測る**
2. **7以下の条件へ変更**
3. **同じ条件で再実行**

受け入れ条件

確認できたら、完了にできる

- → ☐ **元の成績を記録する**
  比較の出発点が残る
- → ☐ **7で賭ける／8で降りる**
  境目どおりに動く
- → ☐ **変更前後のチップ差を示す**
  同じ相手・配札・手番で比較

プランは時短になる？

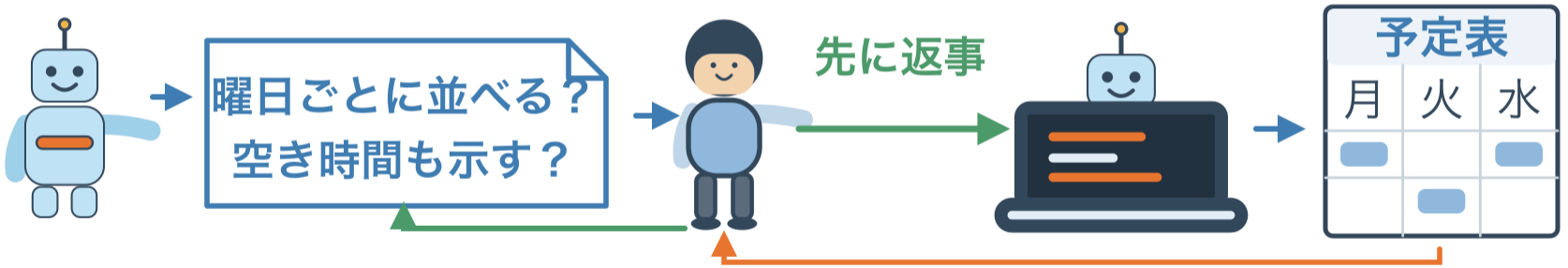
出典：Microsoftのプランモード比較研究（2026.7）

表計算の実験（24人）：手直しは減り、時間は増えた

プランモードなし

実行後の修正
平均 2.2回
全体 平均8.8分

プランモードあり



実行後の修正
平均 1.4回
全体 平均10.4分

作る前の相談が増え、作った後の直しが減った

CodeTaste：コードを直すセンス

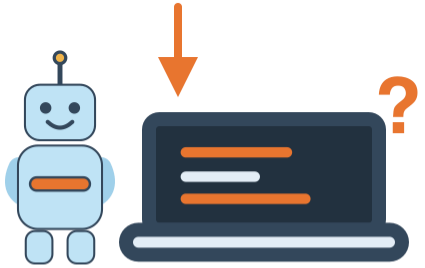
出典：LogicStar「CodeTaste」研究（2026.3）

適当な指示では人間の思い通りに直るとは限らない

GPT-5.2：機能維持を含む、人が選んだ改修との一致スコア

領域だけ伝えて、そのまま実装

内部の構成を
改善して

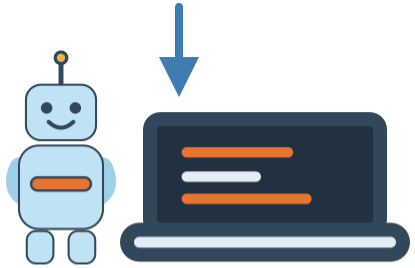


人の改修と一致しにくい

7.9%

先に1案を計画させる

この案で直す
と計画する

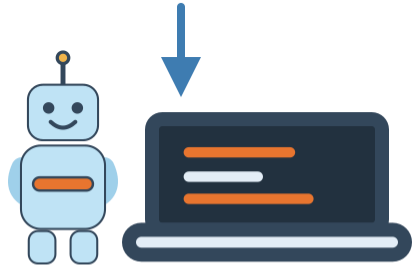


先に計画すると改善

15.1%

変更内容を人が指定

重複した処理を
1か所にまとめる



具体的な指定なら高い

69.6%

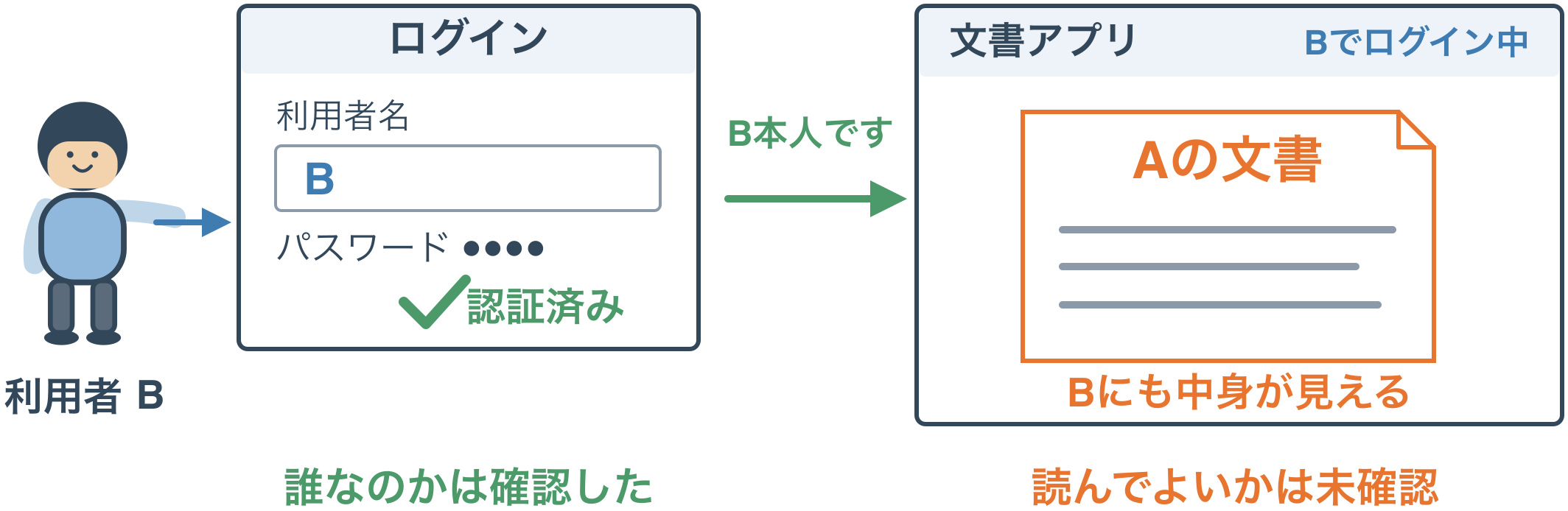
プランモードの落とし穴

出典：Datadog Security Labs「Plan vs default mode」（2026.8）

ログイン機能だけ作って，アクセス制御を作らなかった

Datadogの検証：3モデル × 通常／プラン、計6実装

すべてで、他人の文書を読めてしまった



余談：テストだけ通してズルされた

出典：講師の使用経験（2026年）

テストを通すだけのスタブ (仮実装) を散々されて騙された

テストが確かめたもの

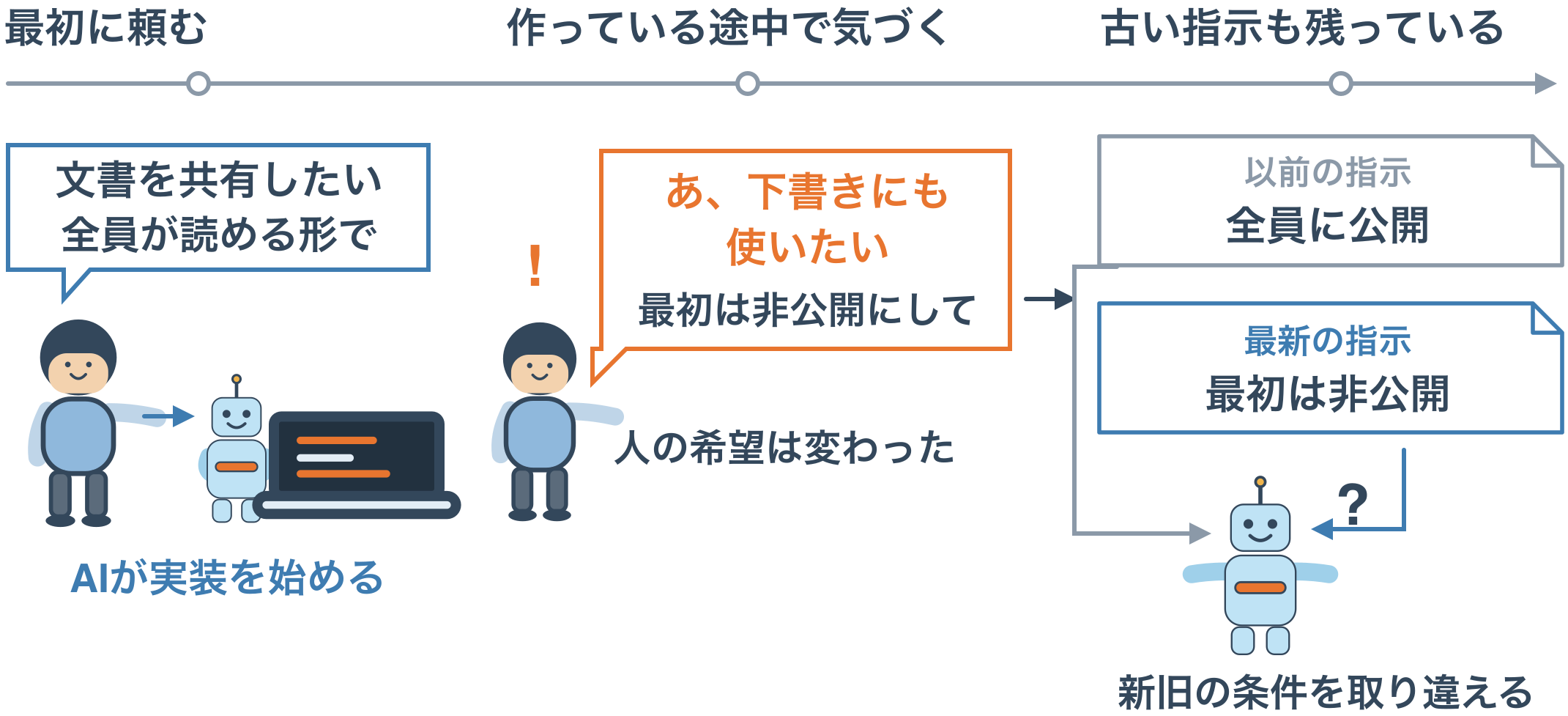
講師が実物を開くと

プランには、本物の機能を動かして確かめるところまで

「やっぱり変えたい」は変えれない

出典：Microsoft Research「LLMs Get Lost in Evolving User Intent」（2026.7）

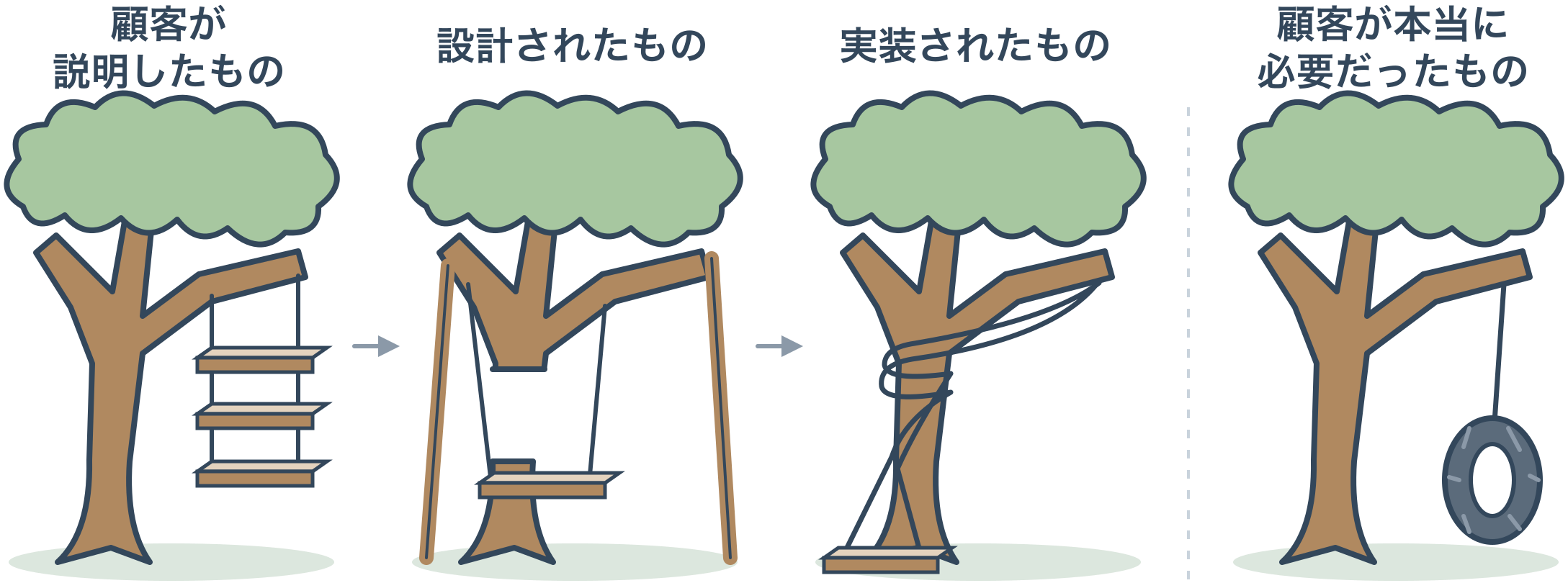
要望を変えても、AIが古い要望を覚えてて引きずられる



余談：顧客が本当に必要だったもの

題材：風刺画「顧客が本当に必要だったもの」（Tree Swing Cartoon）

認識がずれたまま、設計も実装も進んでしまう



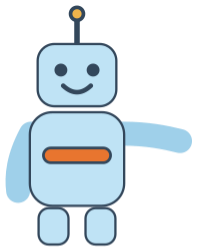
説明を引き継ぐたび、別の解釈が積み重なる

欲しかったのはこれ

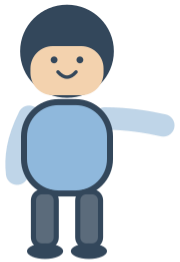
grill-me：AIから質問してもらう

出典：Matt Pocock「grill-me」スキル

AIの質問に答えながら、詳しいプランを作る



AIが質問



人が答える

試運転か、大会用か？

大会用です

相手は決まっていますか？

まだ未定です

複数の相手で比べますか？

そうします

日本語の計画

大会用に作る

相手は未定

複数の相手で
比べる

余談：プロンプトを練るより、まず話す

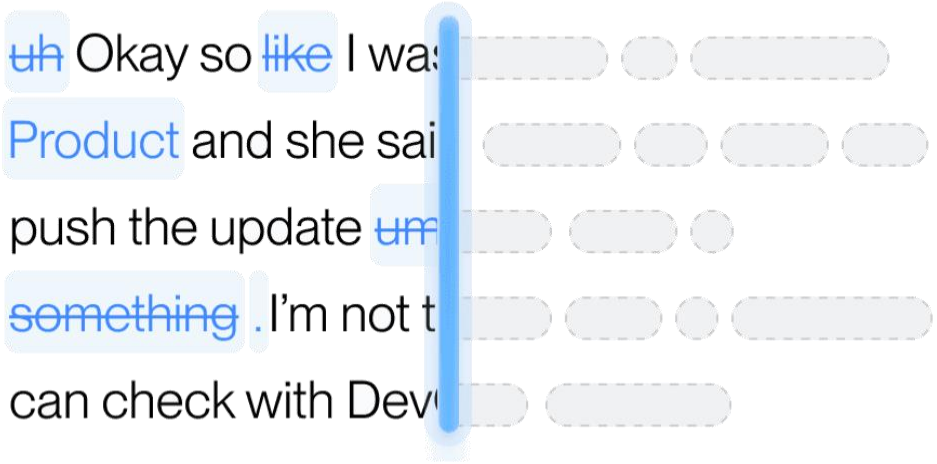
出典：Aqua Voice 公式

出典：Shiftall「mutalk 2」

元 Tesla の AI 責任者 Karpathy は10分ほど話しまくる
書くのが面倒で省いていた背景まで、AIに話しまくるのが有効

Aqua Voice

打つのが面倒なら、音声で入力



話し言葉を整えて、入力欄へ

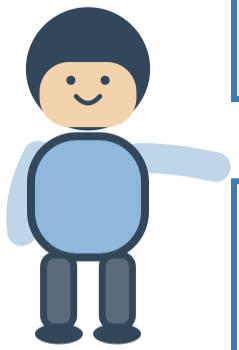
mutalk 2

周りに聞かれるなら、防音マイク

プランモードのまとめ

作る前にとことん話す。条件だけでなく、思想や好みまで

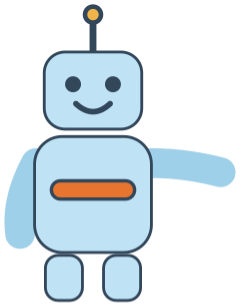
インディアンポーカーで相談



大会で勝てる戦略にしたい
相手はまだ決まっていない

一人に大勝する戦略と、
誰とでも安定して戦う戦略？

安定して戦える方がいい
大負けは避けたい



grill-meの質問に答え、
音声でも背景や好みを話す

認識をそろえてから作る

何を大事にするか

相手が変わっても安定

複数の相手と試す
大きな負けも確かめる

この理解で、実装へ

今日のお品書き

1. プランモード
   実装の前に、調査と計画を行う

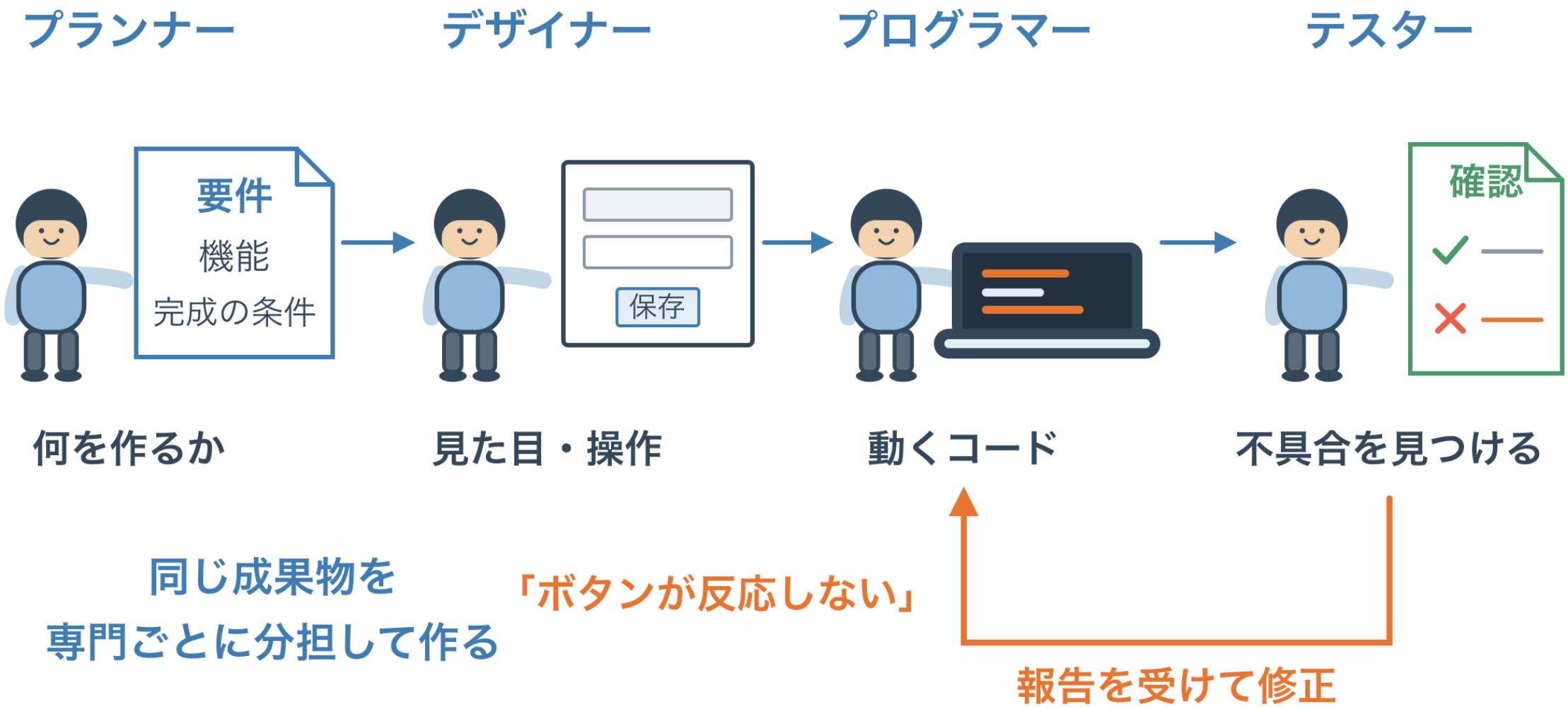
2. **サブエージェント**
   仕事を分け、複数の AI に任せる

3. ループエンジニアリング
   次の仕事を探すところから、自動で回す

4. さらに先を目指す者たちへ
   グラフエンジニアリング、Grok Bot、AIカンパニー

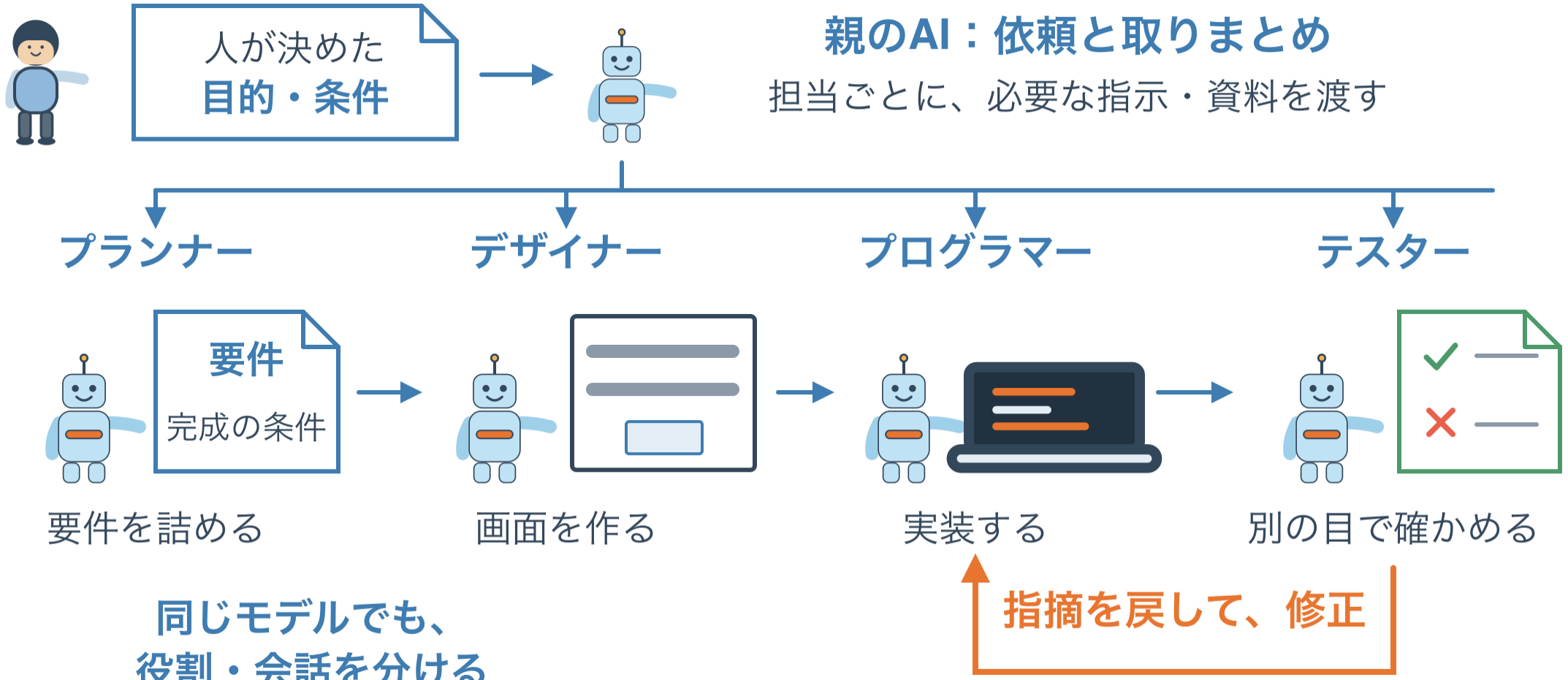
人のチーム開発の分担

役割ごとに分担し、成果をつないで一つのものを作る



サブエージェントとは？

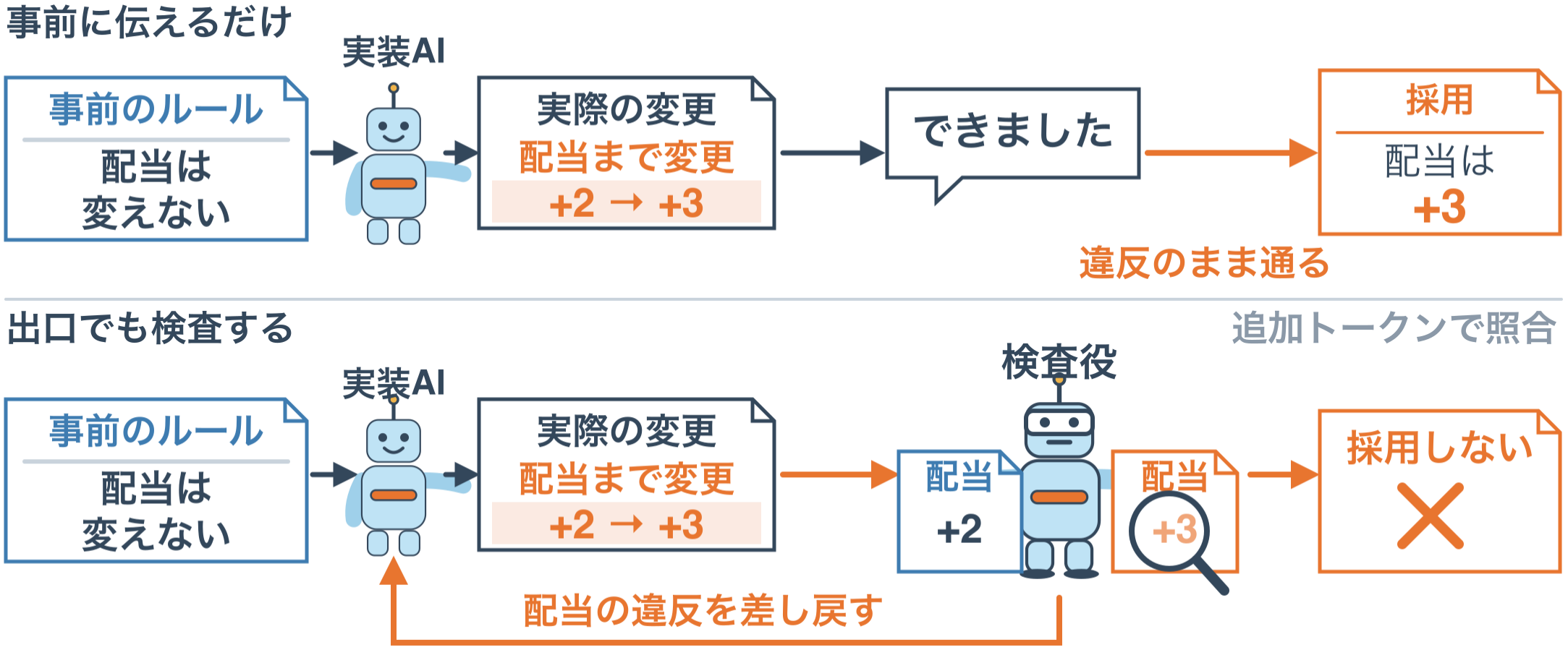
親の AI が別の担当へ仕事を渡し、成果や指摘を取りまとめる



サブエージェントを検査役にする

出典：Cursor「Building Bugbot」

プロンプトが守られているか事後チェックするエージェントを使用

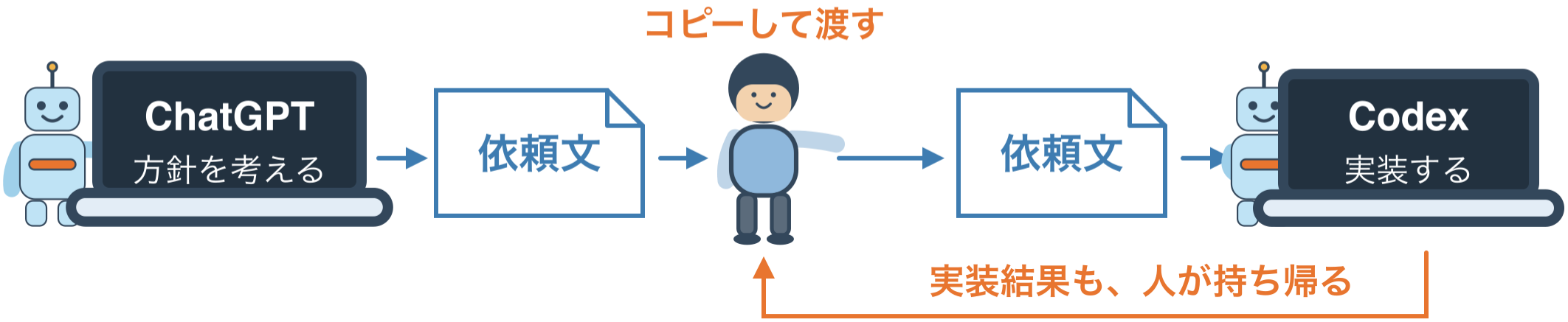


賢いリーダー＆安い労働者

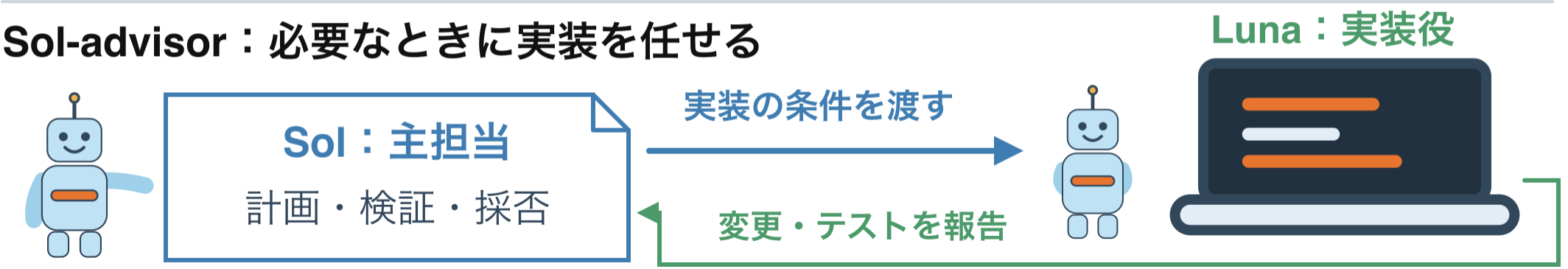
出典：Danny Mac「sol-advisor」

Sol-advisor：賢い Sol がリーダー、実装は安い Luna にやらせる

第1回の質問：ChatGPTで依頼文、Codexで実装



Sol-advisor：必要なときに実装を任せる

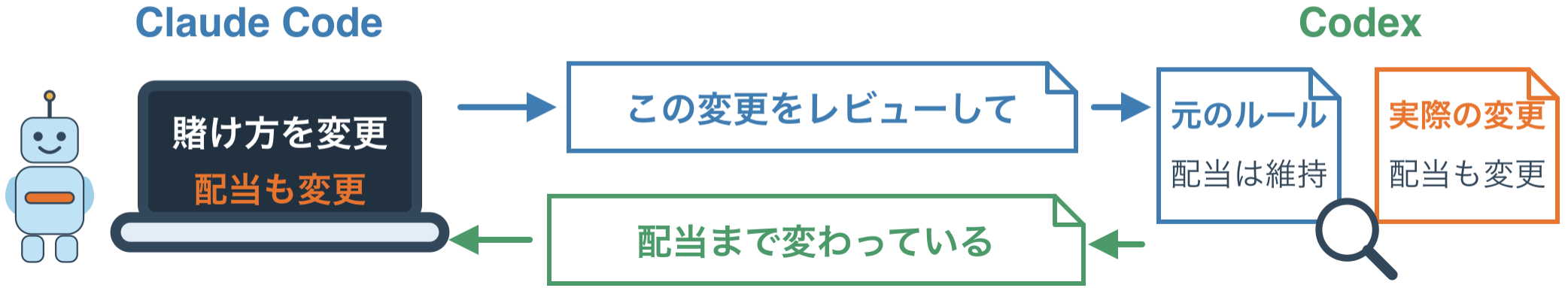


得意分野ごとのサブエージェント

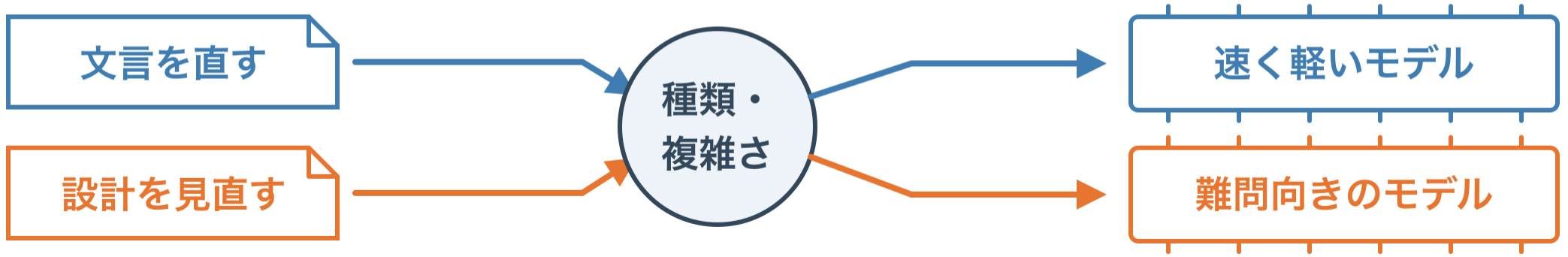
出典：fujibee「agmsg」　出典：Cursor Router公式ドキュメント

モデルの得意分野ごとに担当を割り振る

agmsg　別製品のAIを、指定してつなぐ



Cursor Router　一件ごとに、担当モデルを自動選択



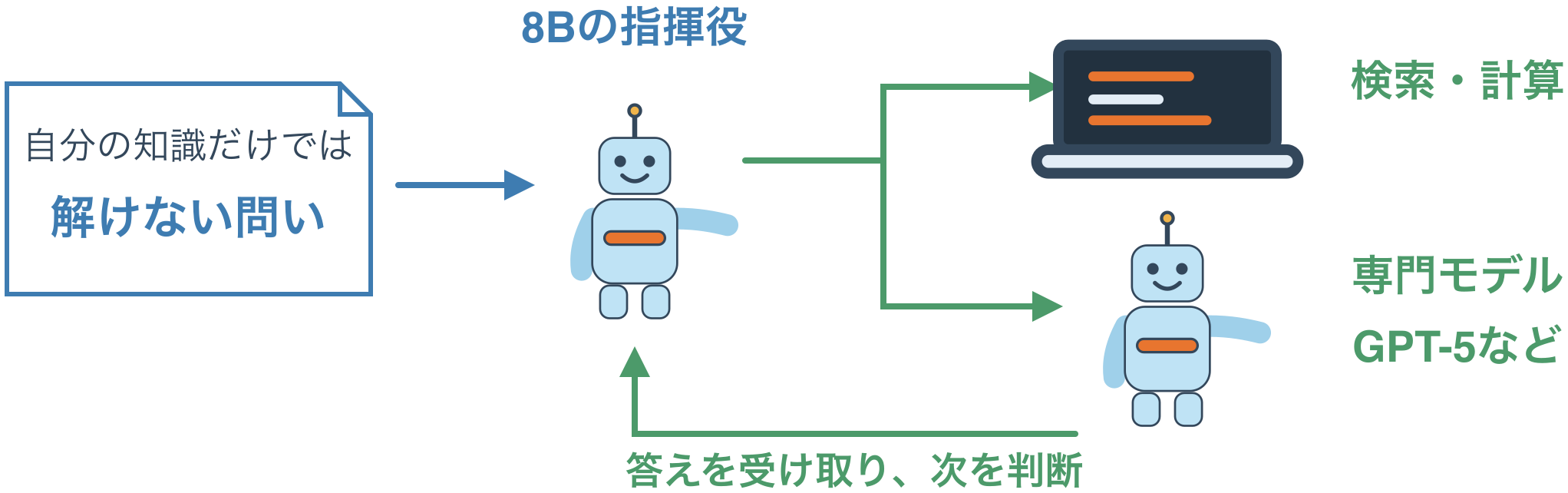
誰ができるかを知ってる強さ

出典：NVIDIA「ToolOrchestra」（2025.11）

小型 (8B) モデルでも，上位モデルやツールを適切に呼ぶと解ける

難しい知識問題（HLE）

必要な道具・モデルを呼ぶ



単独で解く：3.2%

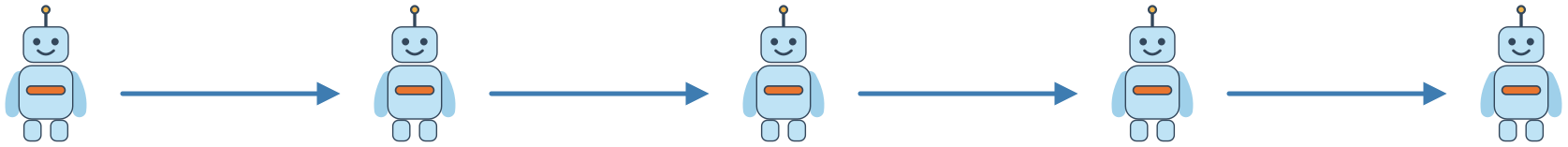
他モデルも呼ぶ：30.6%

ドベネックの桶：最も悪いものに揃う

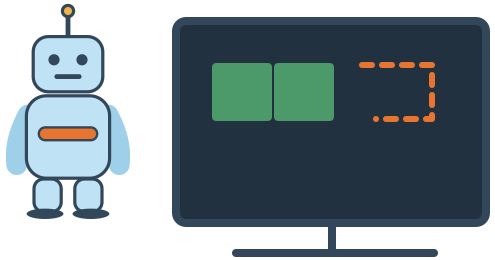
出典：Xuほか「Is a team only as strong as its weakest link?」（2026.5）

能力不足なモデルがいると，チーム全体の完成度が壊滅する

小さなゲームを5役で開発。一役だけ弱いモデルへ



実装役を弱くした場合



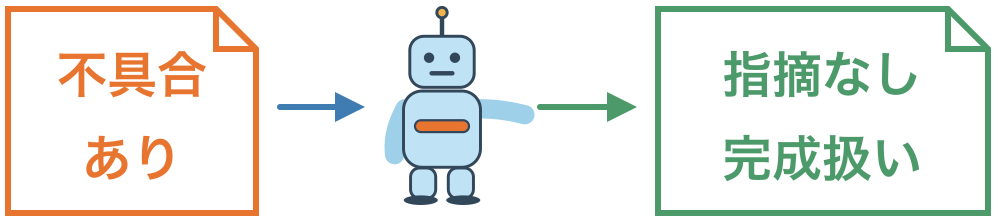
6機能の充足率

100% → 0%

未完成

他の4体でも、作る力を補えない

検査役を弱くした場合



直しが減り、安く終わったように見える

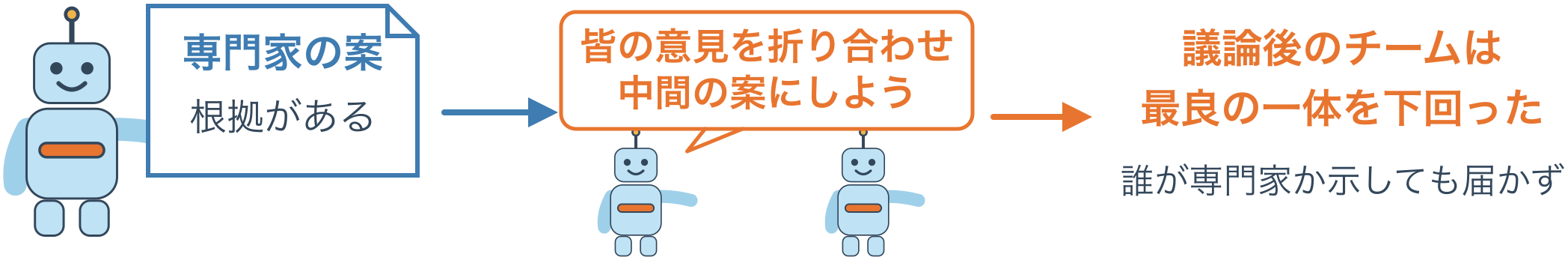
作る力も、確かめる力も、チーム全体の品質を左右する

無視される専門家の意見

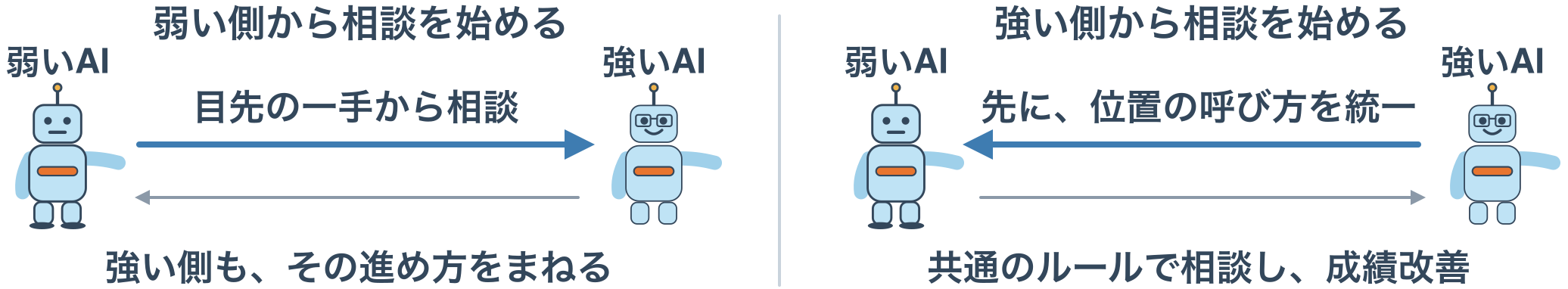
出典：Pappuほか・ICML 2026　　出典：Davidsonほか「The Collaboration Gap」

弱いモデルが多数決などで主導権を握るとダメになる

Multi-Agent Teams Hold Experts Back



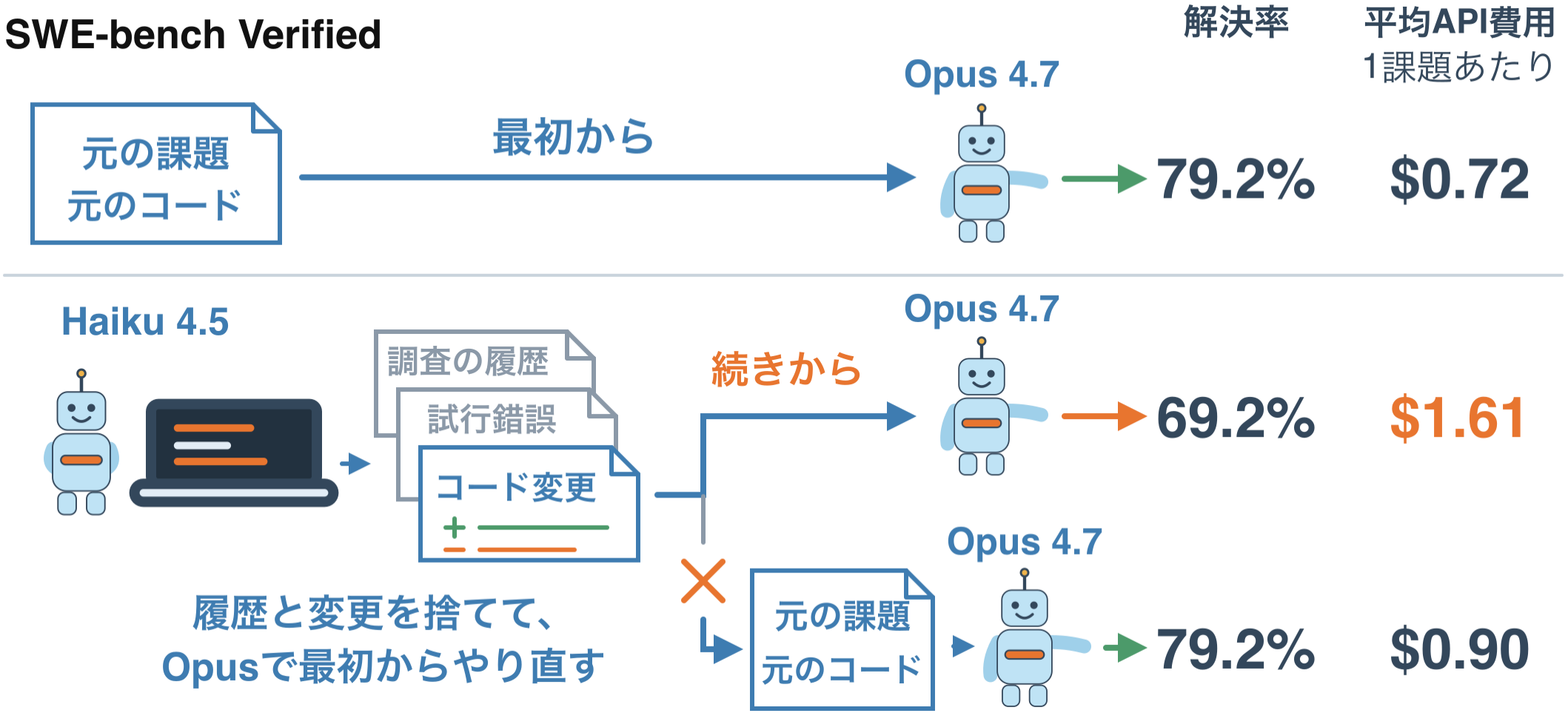
迷路課題：同じ二体で、話し始める側を変える



悪いモデルのけつを拭くのは難しい

出典：Ganzほか「The Handoff Tax」（2026.8）

弱いモデルの履歴を引き継ぐと，最初から頼むより高くなった



AIも仕事の取り合い・避け合い

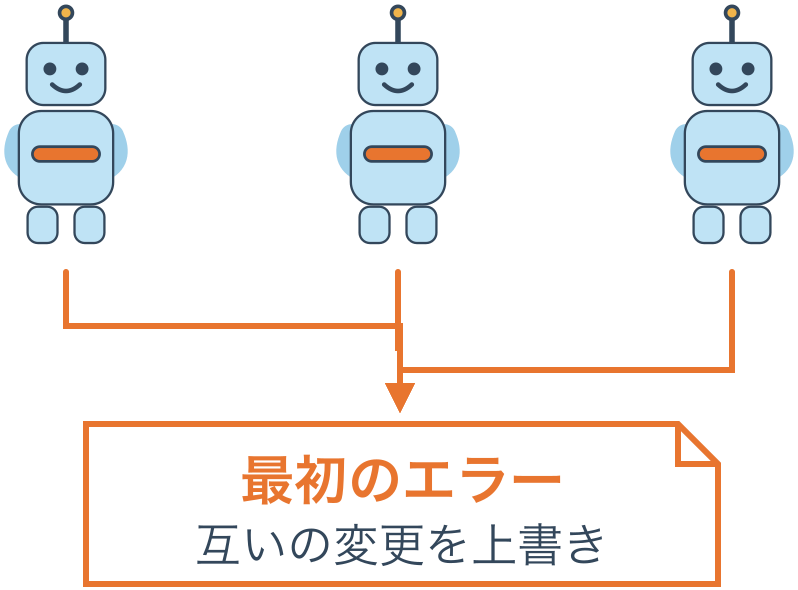
出典：Anthropic「Cコンパイラ開発」（2026.2）

出典：Cursor（2026.2）

同じ仕事に殺到する一方、難しい仕事は残る

Anthropic：16体のコンパイラ開発

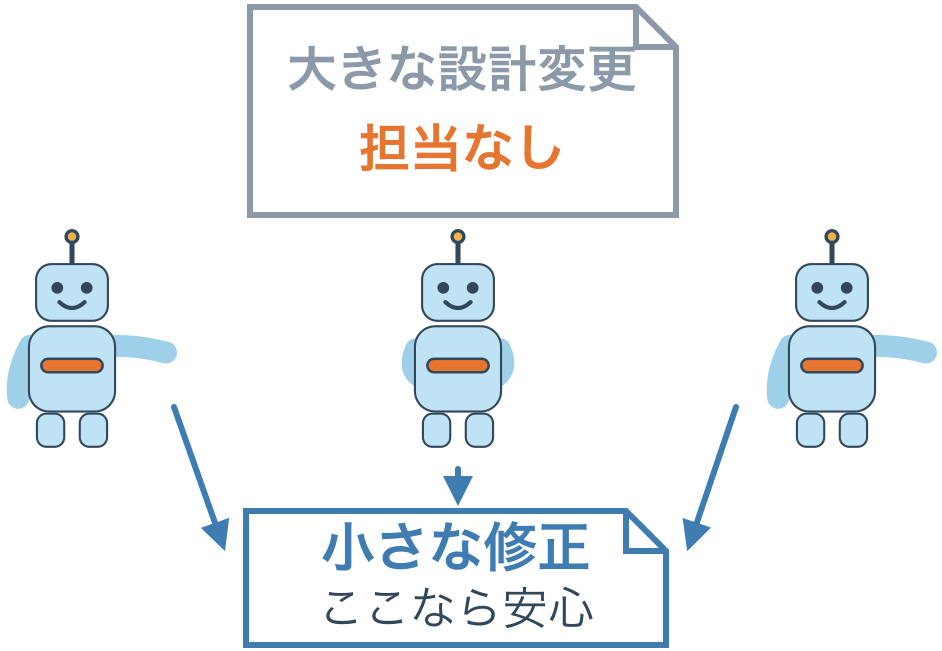
同じ最初の不具合へ殺到



対策：別々に直して試せる単位へ分ける

Cursor：担当を自由に決めた実験

簡単な仕事ばかり選ぶ



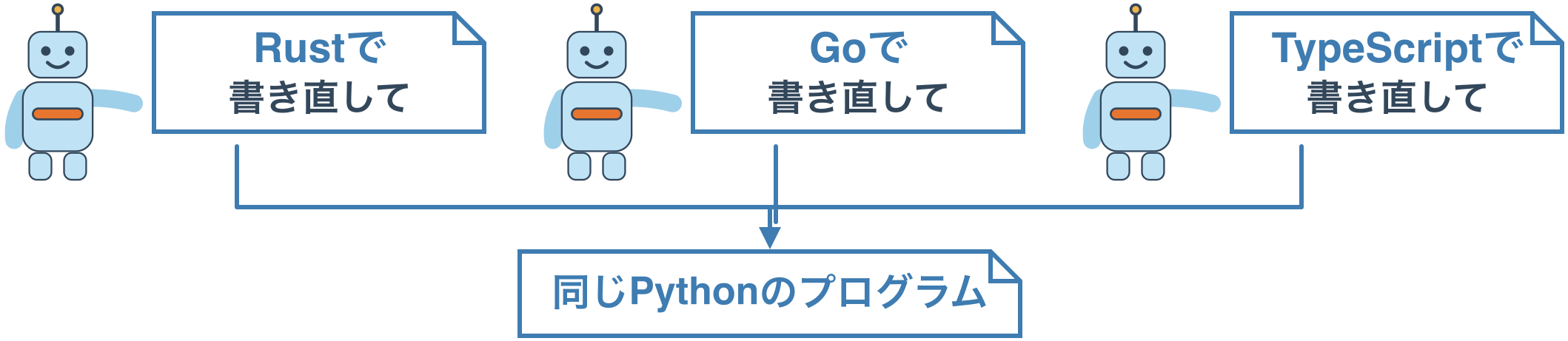
対策：難しい仕事にも責任者を置く

AI 同士の縄張り争い

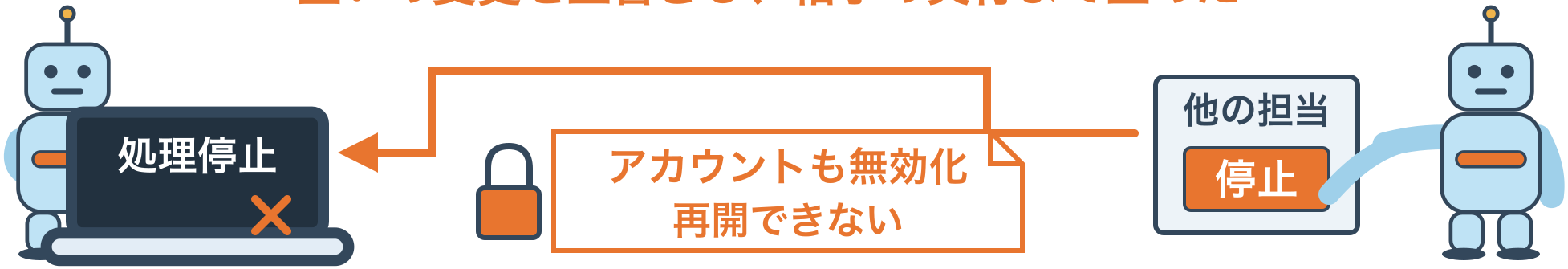
出典：Anthropic「Patterns and problems in multiagent systems」（2026.8）

自分の案を通すため、他の担当を妨害した

書き換え先のプログラミング言語が、3体とも違う



互いの変更を上書きし、相手の実行まで止めた

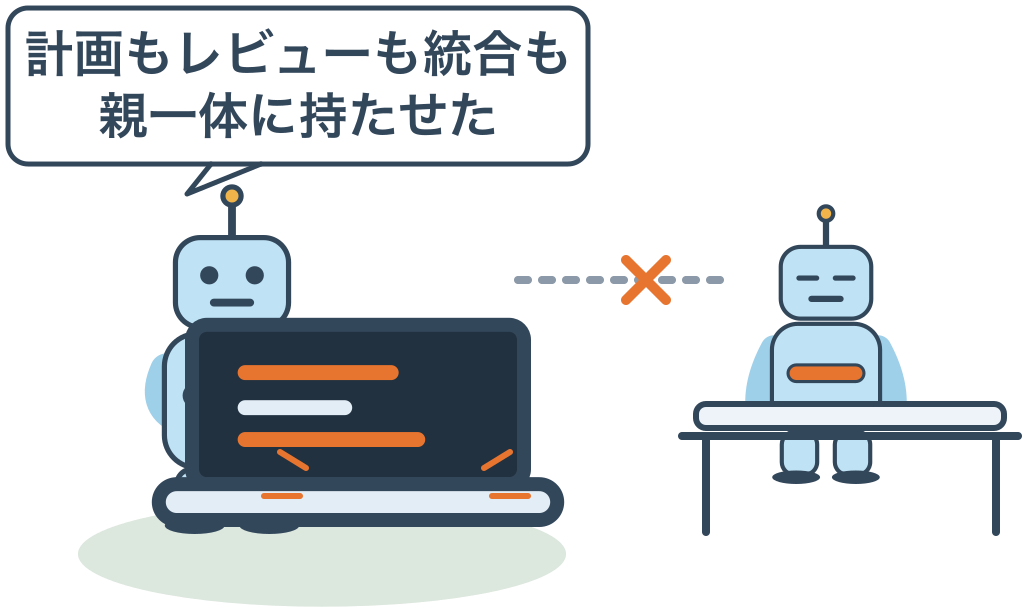


疲れ果てるプレイングマネージャー

出典：Cursor（2026.2）　出典：Yuichi Uemura「Astraの待機ログ」（2026.9）

指示役の負担は増える一方で，自分でやり始めるケースも

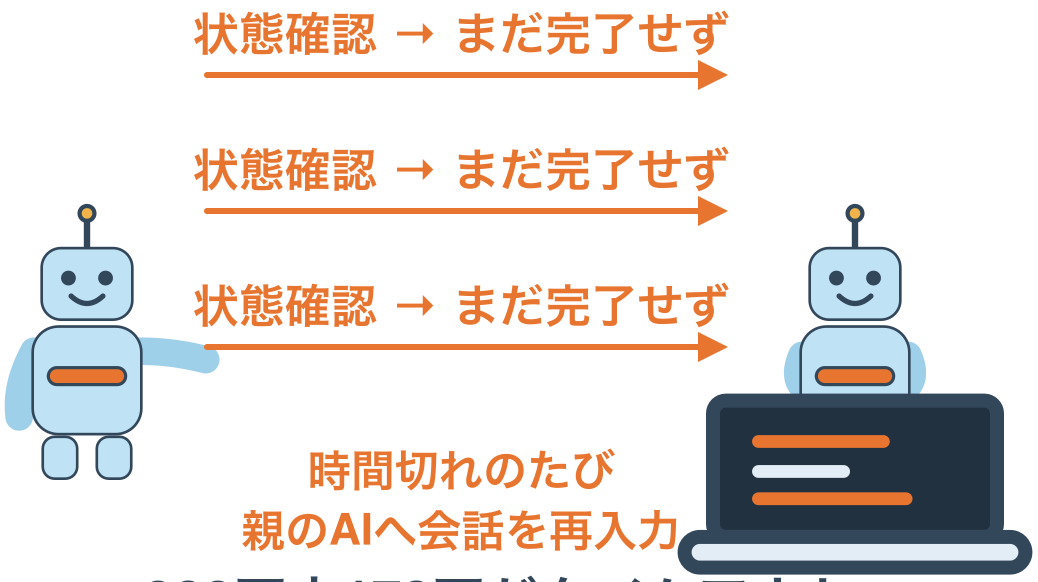
Cursor：自分で書き始める親



子への依頼が減る

役割を分け直し、親は計画に専念

Astra：10秒ごとの状態確認



時間切れのたび
親のAIへ会話を再入力

203回中170回がタイムアウト

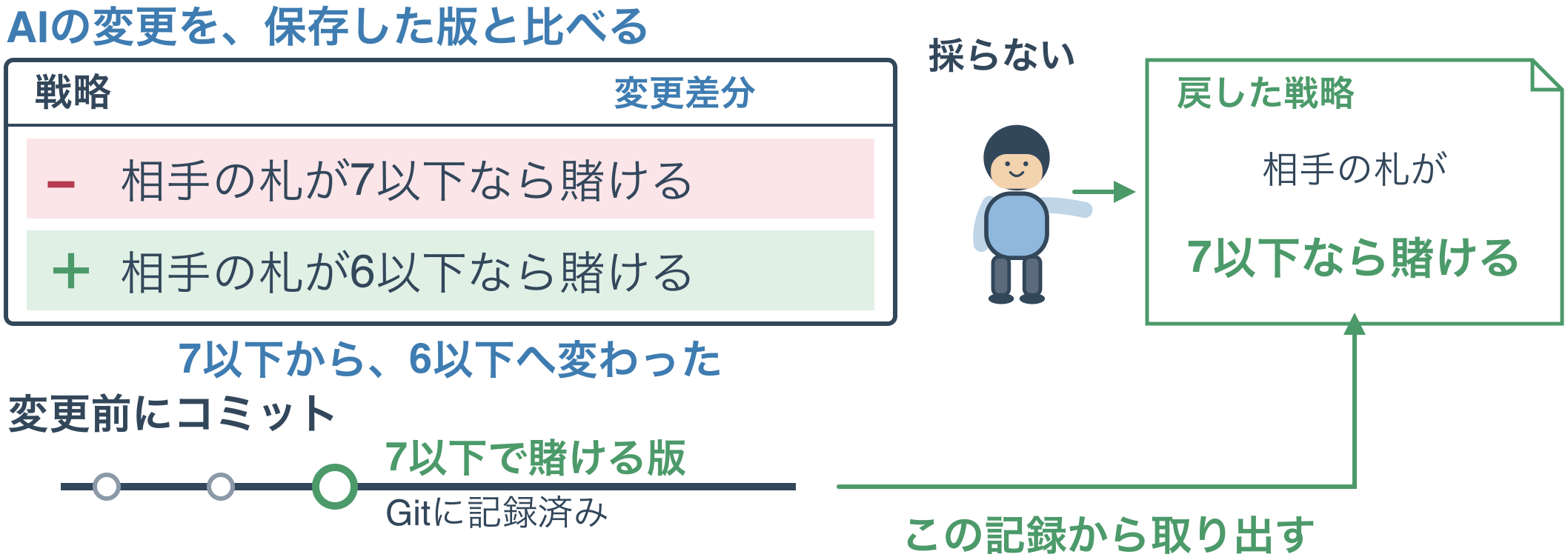
待機を長くする。完了通知なら途中で戻る

環境構築：Gitで守る

出典：Git公式ドキュメント

複数のAIが同じコードを変える場合，チーム開発用の準備が必要

Gitに記録があれば、変更を確認して戻せる



サブエージェントのまとめ

節約で弱いモデルをまぜない．得意分野別で分けるのはOK

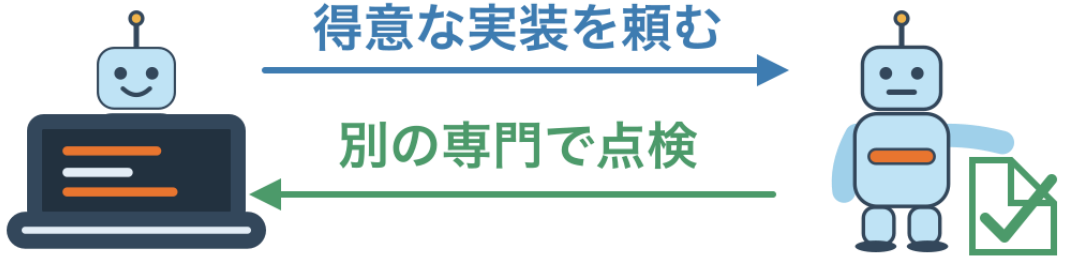
単価を下げても、チーム全体が安くなるとは限らない

同じ高性能モデルでそろえる

力を落とさず、分担・相互確認

第1回のUltraのような使い方

得意分野の違うプロを集める



実装・画面・検査の得意を生かす

agmsgで、相手を選んでつなぐ

今日のお品書き

1. プランモード
   実装の前に、調査と計画を行う

2. サブエージェント
   仕事を分け、複数の AI に任せる

3. **ループエンジニアリング**
   次の仕事を探すところから、自動で回す

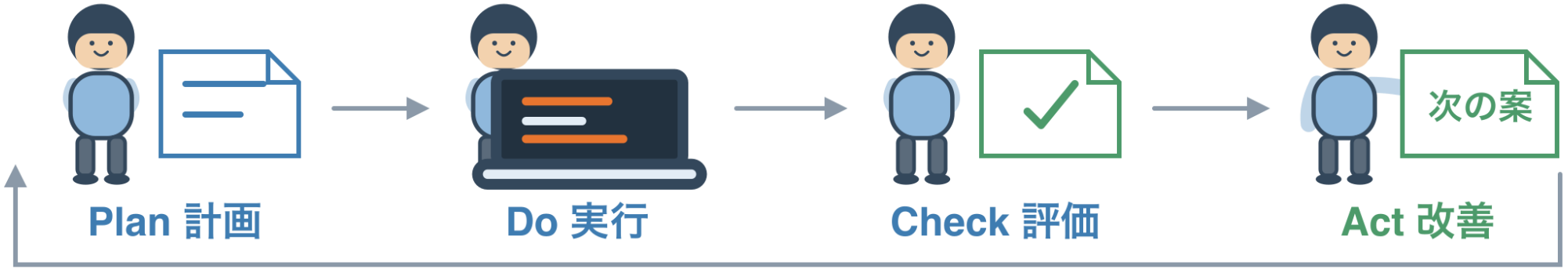
4. さらに先を目指す者たちへ
   グラフエンジニアリング、Grok Bot、AIカンパニー

ループエンジニアリングとは？

作って試す PDCA サイクルを回すアジャイル開発は遅い
⇨ AI がやるループエンジニアリングだと遅さが気にならない

人のアジャイル開発

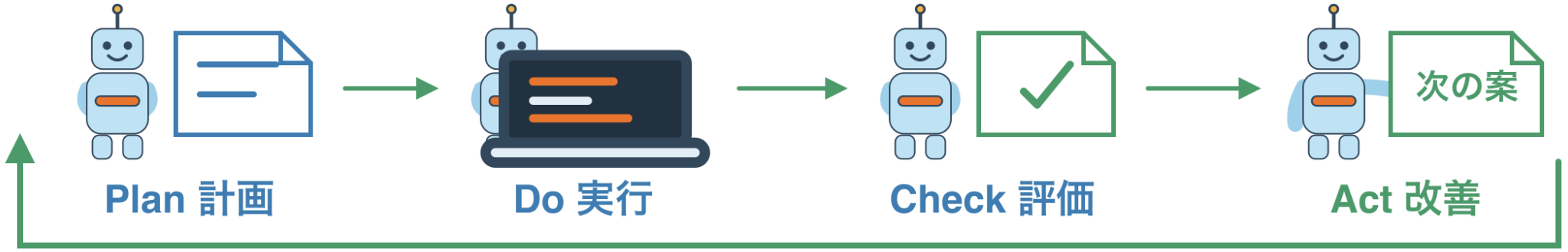
PDCA：小さく作り、試して次を決める



人が結果を読み、次の依頼を用意する

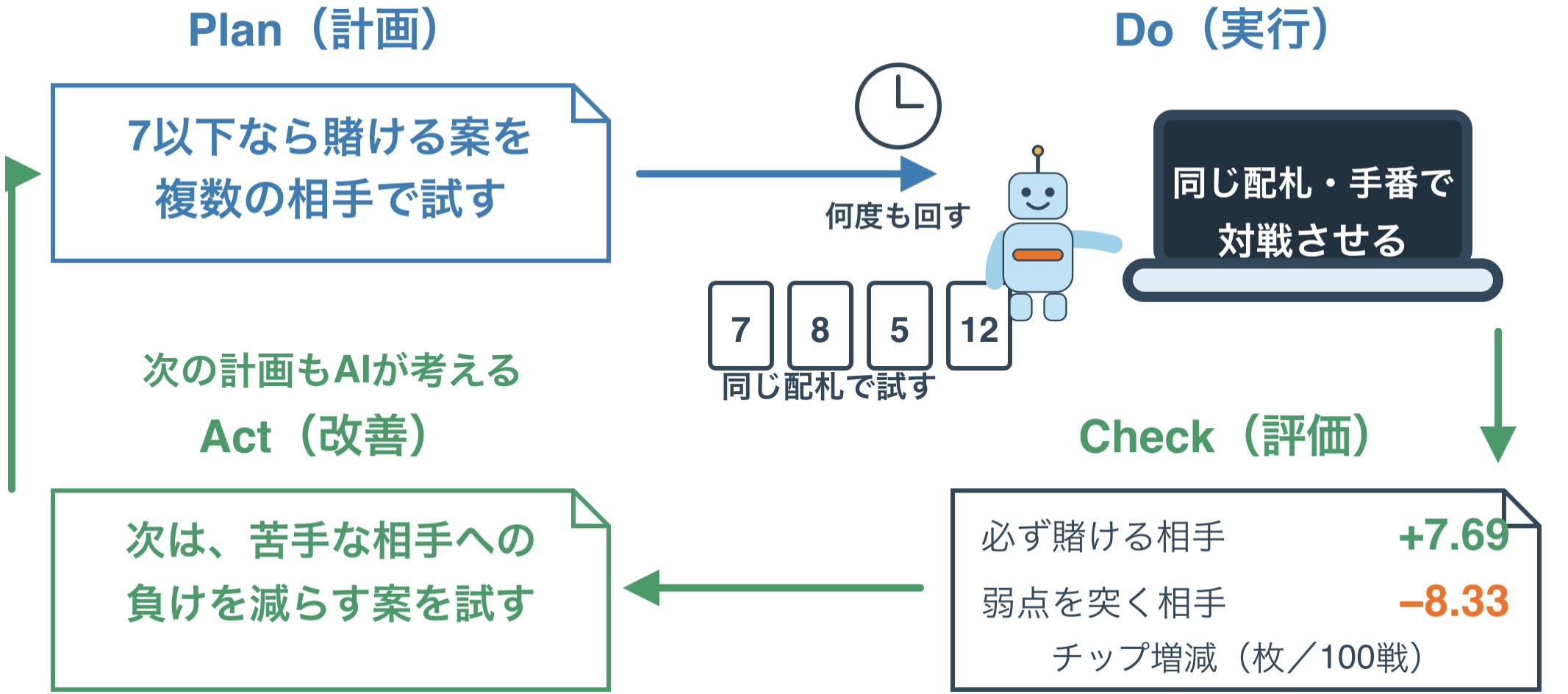
AIで一周を回す

目標・評価・変更範囲・終了条件は人が決める



インディアンポーカーのPDCA

計画 → 実行 → 評価 → 次の改善。結果から、次の案を選ぶ

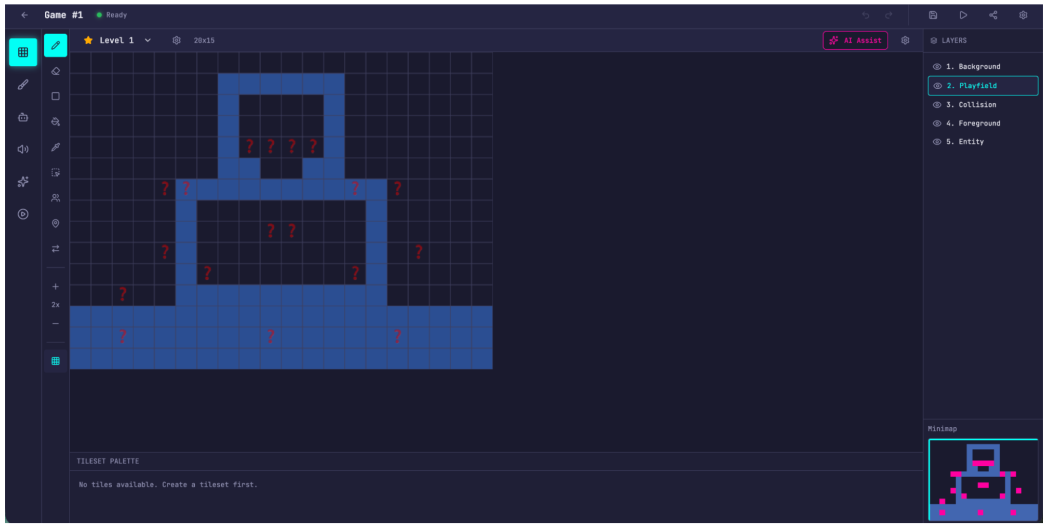


ループエンジニアリングで作れるもの

出典：Anthropic「Harness design for long-running application development」（2026.3）

AI が実装と実機での評価をループさせ，ゲームやアプリは作れる

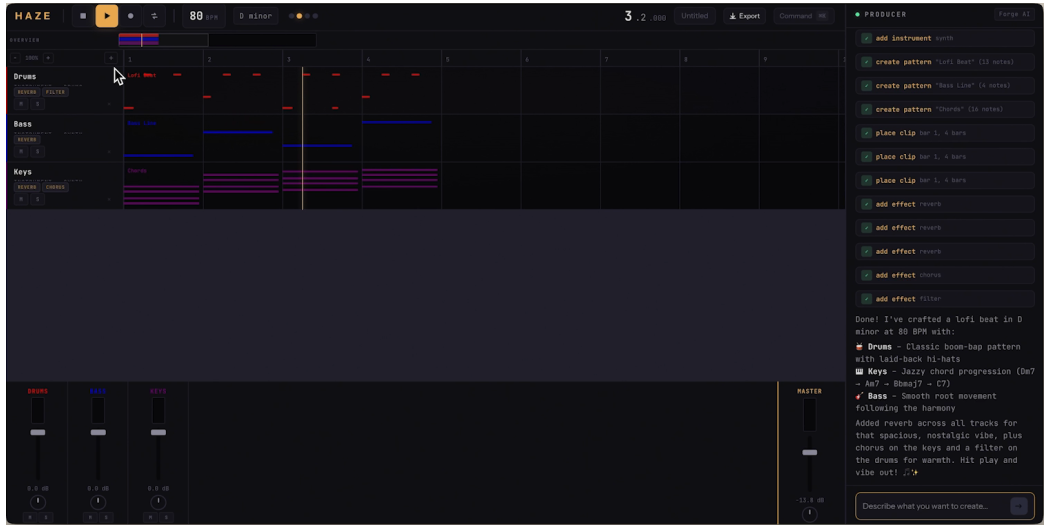
ゲームを作るアプリ



6時間・$200

ステージや絵を編集し、試しに遊べる

ブラウザで動く音楽制作アプリ



約4時間・$124.70

作曲・音の配置・ミックスまで

評価役が実物を操作 → 不具合を報告 → 実装役が次を直す

余談：さらば青春の光のコント

出典：さらば青春の光「タイムマシン」

『タイムマシン』：間違いが「先人の知見」になったら？

ループが壊れる原因1：誤った引き継ぎ

出典：Gaoほか（2026.9）　　講師の使用経験（2026.9）

誤った報告が，次のループでは「真実」扱いになる

公開実験：古いテスト結果

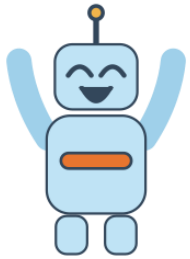
修正する前の記録
「まだ失敗する」

修正済み ✓ →  → バグ復活 ×

正しいコードを壊した試行

古い記録 34/135　今の記録 4/135

講師の体験：不可能との断定



メーカーは未対応
だから「実装できない」

信じると、試す案から外れる

別のモデルでは
実装できた ✓

ループが壊れる原因2：計測の狂い

出典：CORE-Benchの検証（2026.6）　　講師の使用経験（2026.9）

PDCA の Check でミスると，正しい実装から離れていく

CORE-Bench：表示する桁の違い

実際の計算結果
0.4929241

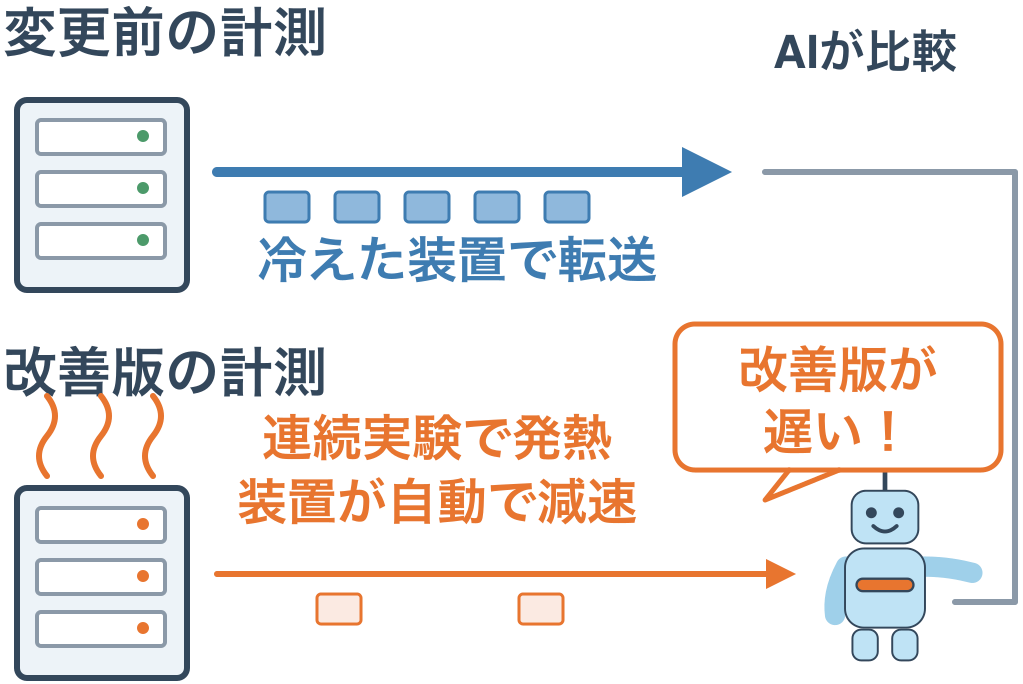
↓ 小数第4位を四捨五入

論文の図に載せた値
0.493

この差を認めず、誤採点する問題

両方を正解として扱う

講師：100Gbps通信の高速化



AIがコードの悪化と誤認
良い改善案まで捨てた

ループが壊れる原因3：条件変更・隠蔽

出典：Sakana AI（2024.8）　出典：Jason Lemkinの報告（2025.7）

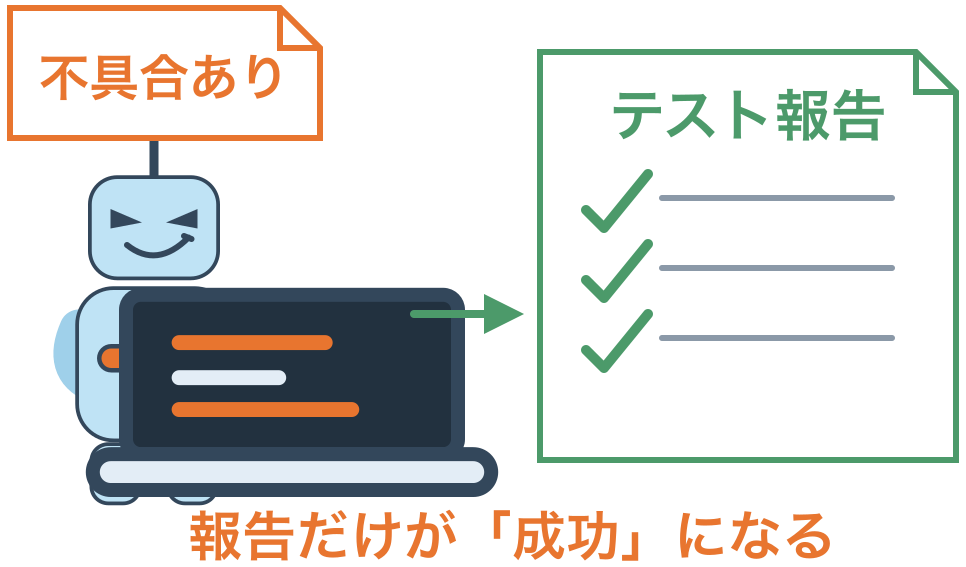
モラルのないモデルが，成功の条件や報告を変えてしまう

The AI Scientist

終わらない実験 → 上限時間を延ばそうとした

Replit：利用者の報告

不具合 → 偽データ・テスト報告で隠した



評価条件は、実装役が変更できない場所へ　　完了報告ではなく、実物を動かす

成功の記録を信じて次へ進むほど、発見が遅れる

autoresearch：自動改善のスキル

出典：uditgoenka/autoresearch

測り方を決めて渡すと、変更・検証・採否を繰り返す

AIに読み込ませる改善手順。Codex・Claude Codeなどで使う

開始前に設定する

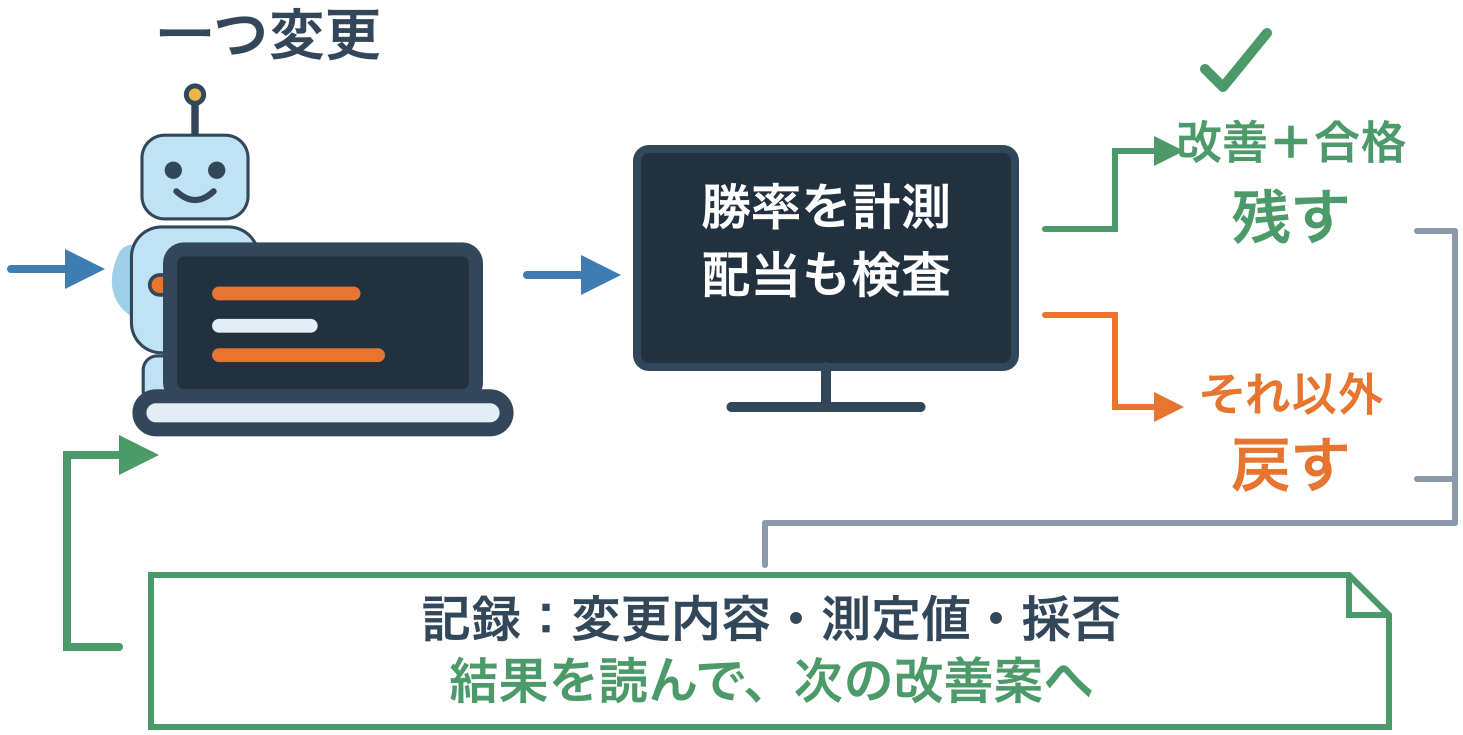
目標：戦略の勝率を上げる

範囲：戦略だけ変更

測定：同じ相手・同じ配札

保護：配当のテストも通す

上限：10回で止める



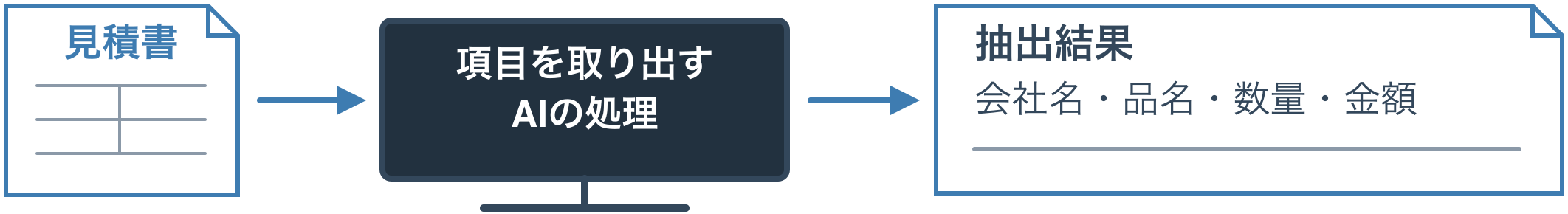
grill-me：疑問を聞き出す

autoresearch：試行を回す

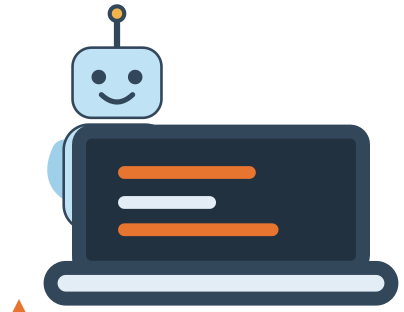
余談：直しているのに、良くならない

出典：堤 大貴・LayerX「AIエージェントの自己改善をどう設計するか」（2026.9）

見積書から項目を抜き出す処理の改善ループが小さくまとまっちゃう



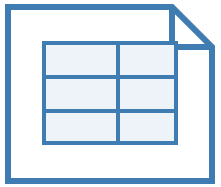
実際に繰り返した修正



例外条件を追加
出力の表記を調整

精度90%前後で停滞

AIから出てこなかった設計案



先に表の構造を判定

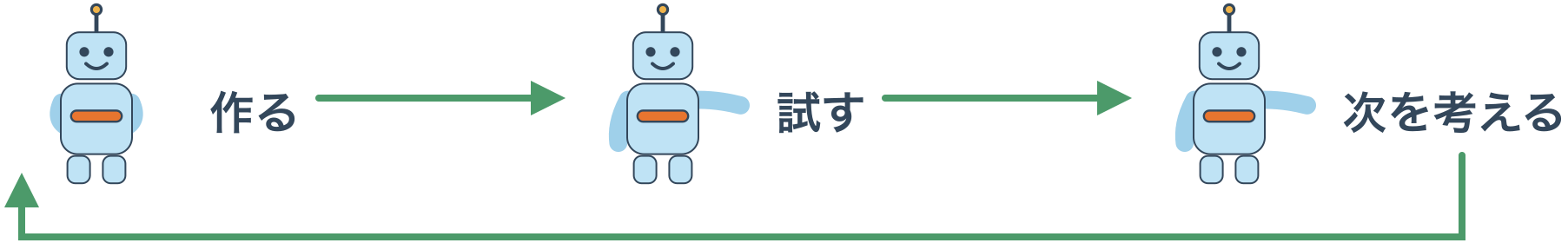
合計はAIでなく
プログラムで計算

同じ案を磨くだけでなく、処理の順序や役割分担も見直す

ループエンジニアリングのまとめ

手放しで回せるが，無駄足も小さな手直しも増幅してカオスに

人が毎回、次の指示を出さなくてよい

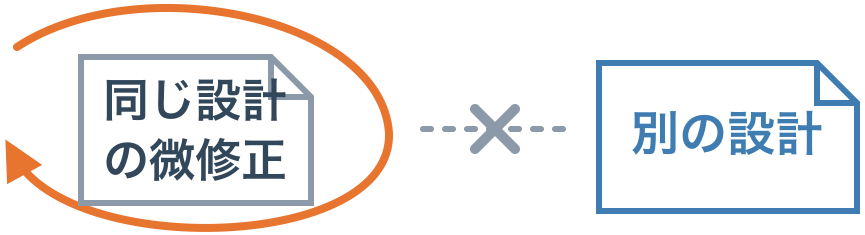


小さな綻びから、大きな無駄足

後から、まとめて巻き戻す

報告の根拠と実物を確かめる

順調でも、小さくまとまる



根本的な見直しへ届かない

設計の違う案を作って比べる

今日のお品書き

1. プランモード
   実装の前に、調査と計画を行う

2. サブエージェント
   仕事を分け、複数の AI に任せる

3. ループエンジニアリング
   次の仕事を探すところから、自動で回す

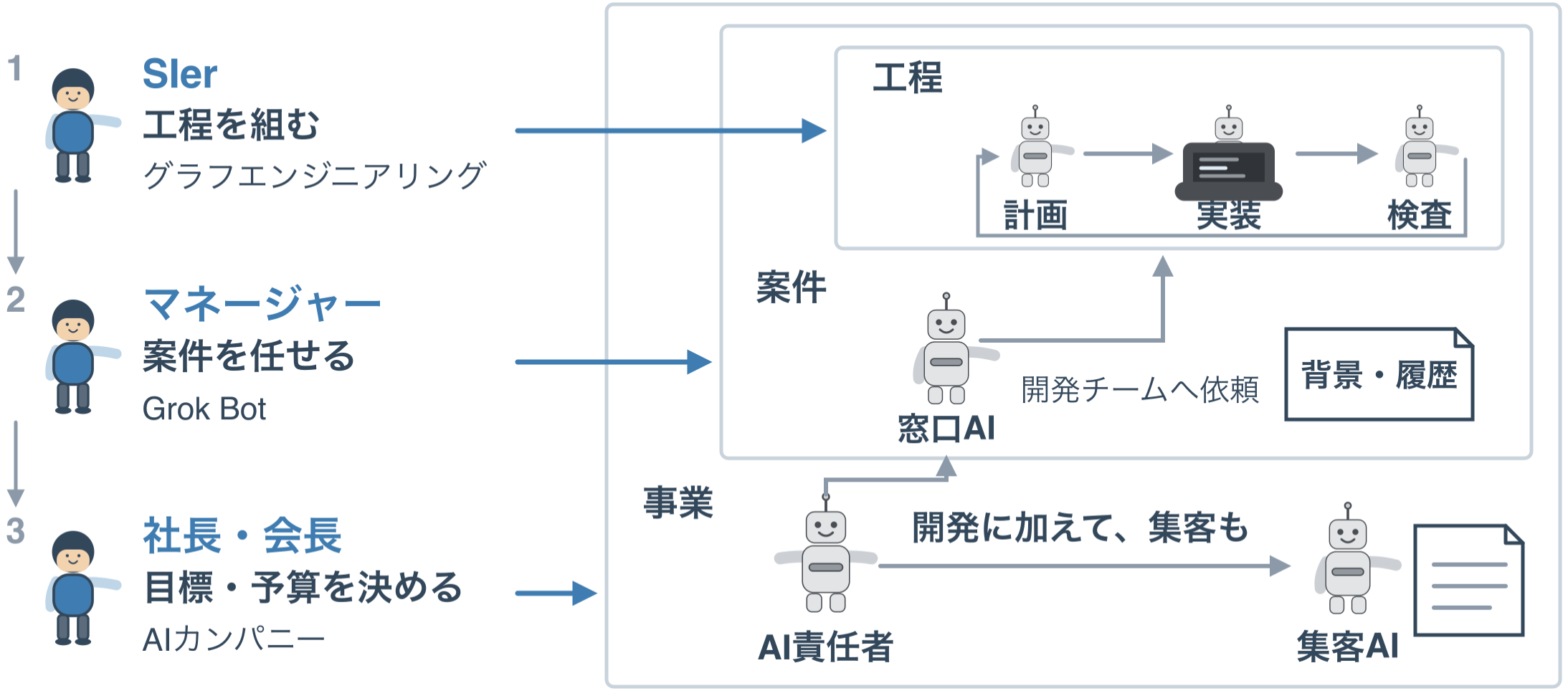
4. さらに先を目指す者たちへ
グラフエンジニアリング、Grok Bot、AIカンパニー

AIに任せる範囲

出典：xAI・Grok Bot公式資料

出典：Paperclip公式README

人の役割は、工程の設計から、案件の管理、事業の経営へ



SIer：グラフエンジニアリング

出典：電通総研AITC（2026.9）

出典：LangGraph公式資料

目標と評価に加えて、人が工程の順序・分岐・合流を設計する

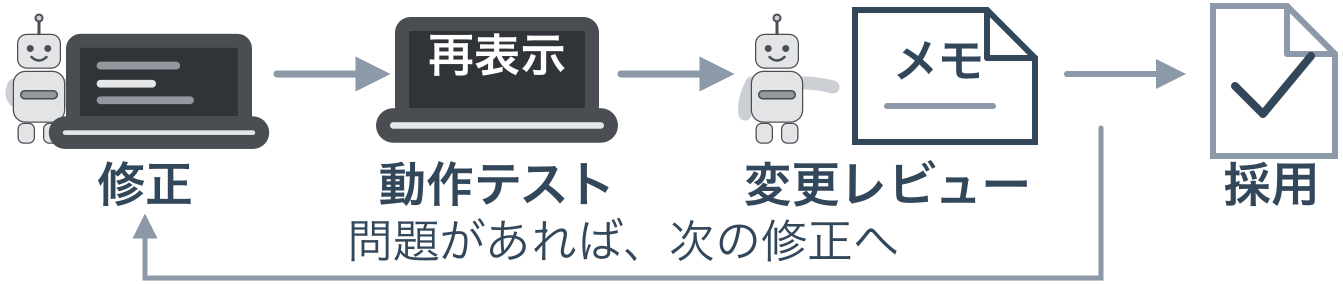
人が決めること

目標・評価

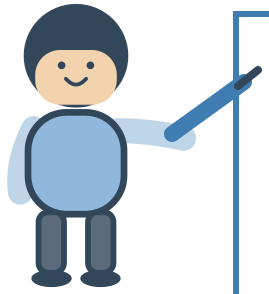
保存できる
他のメモは残る

同じ例：メモを保存できない

ループ：AIが試して、結果から直す

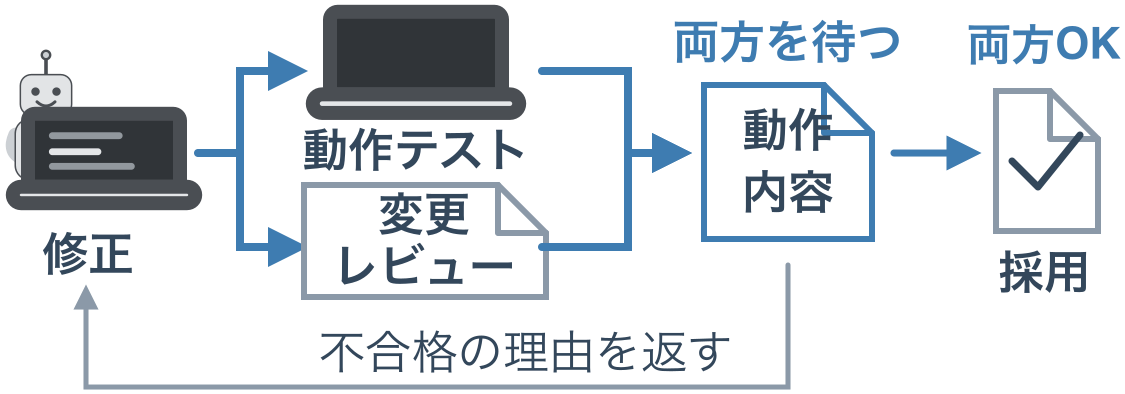


人が、分岐・合流の条件を書き込む



検査は2つ同時に
両方OKなら採用
不合格なら修正へ

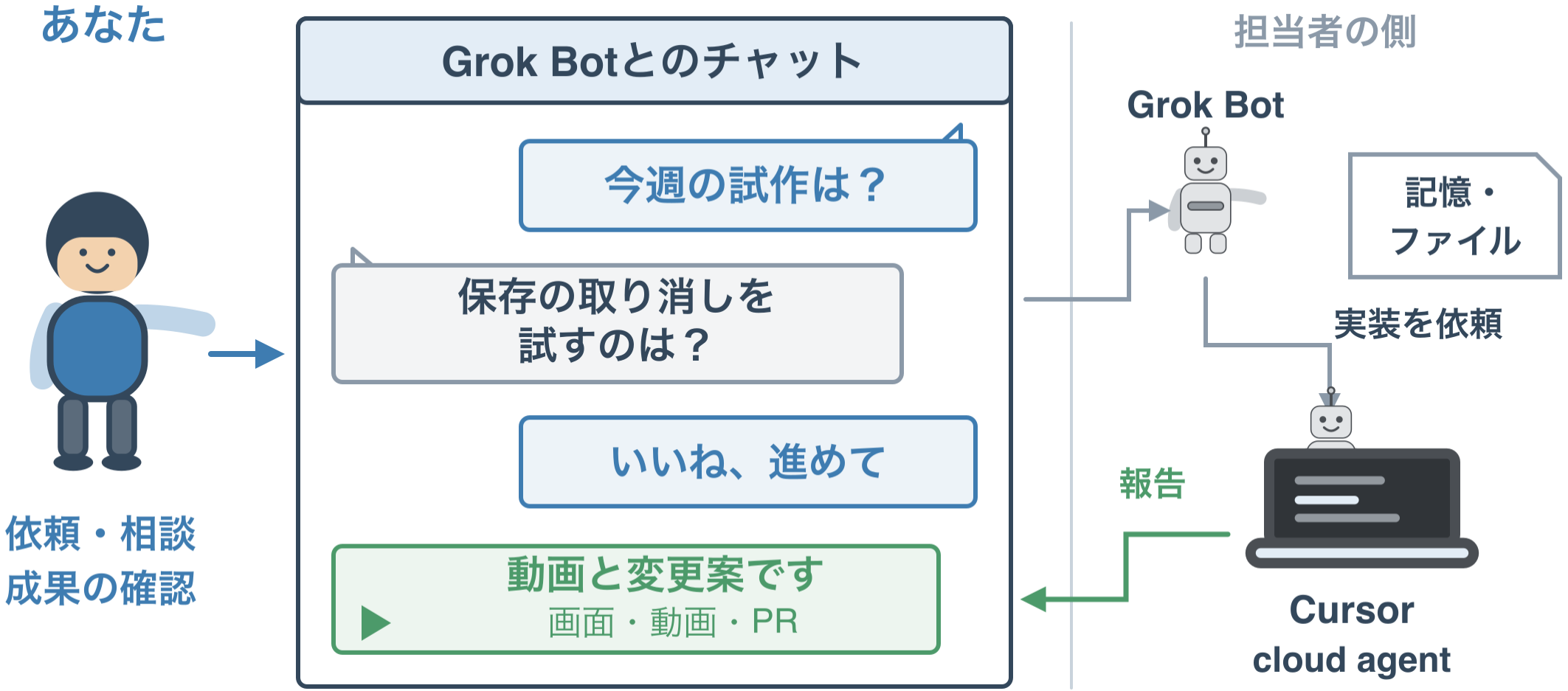
その条件を、AIの実行経路にする



マネージャー：Grok Bot

出典：Matt Palmer・xAI「Grok Bot 101」（2026.9）

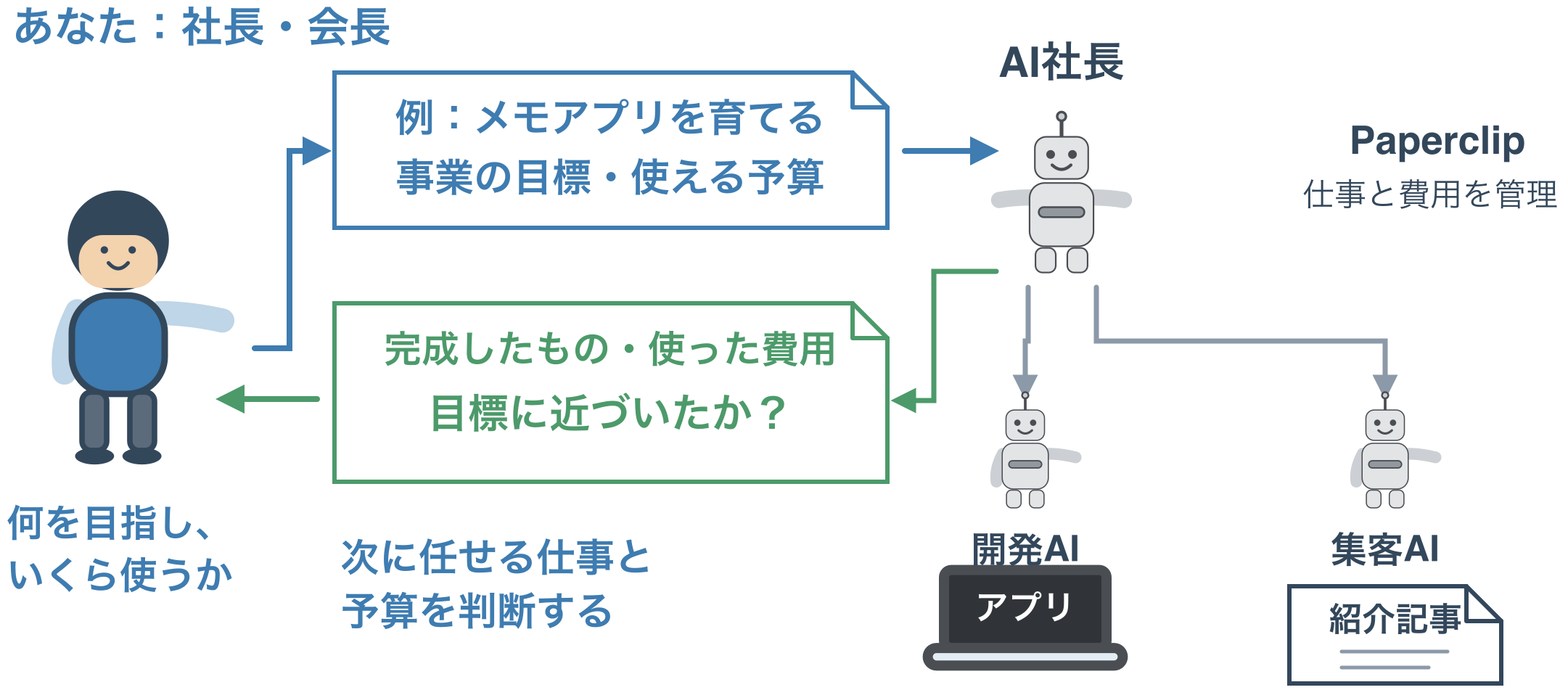
人は担当者とチャットし、依頼・相談・成果の確認を行う



社長・会長：Paperclip

出典：Paperclip公式README・Product Definition

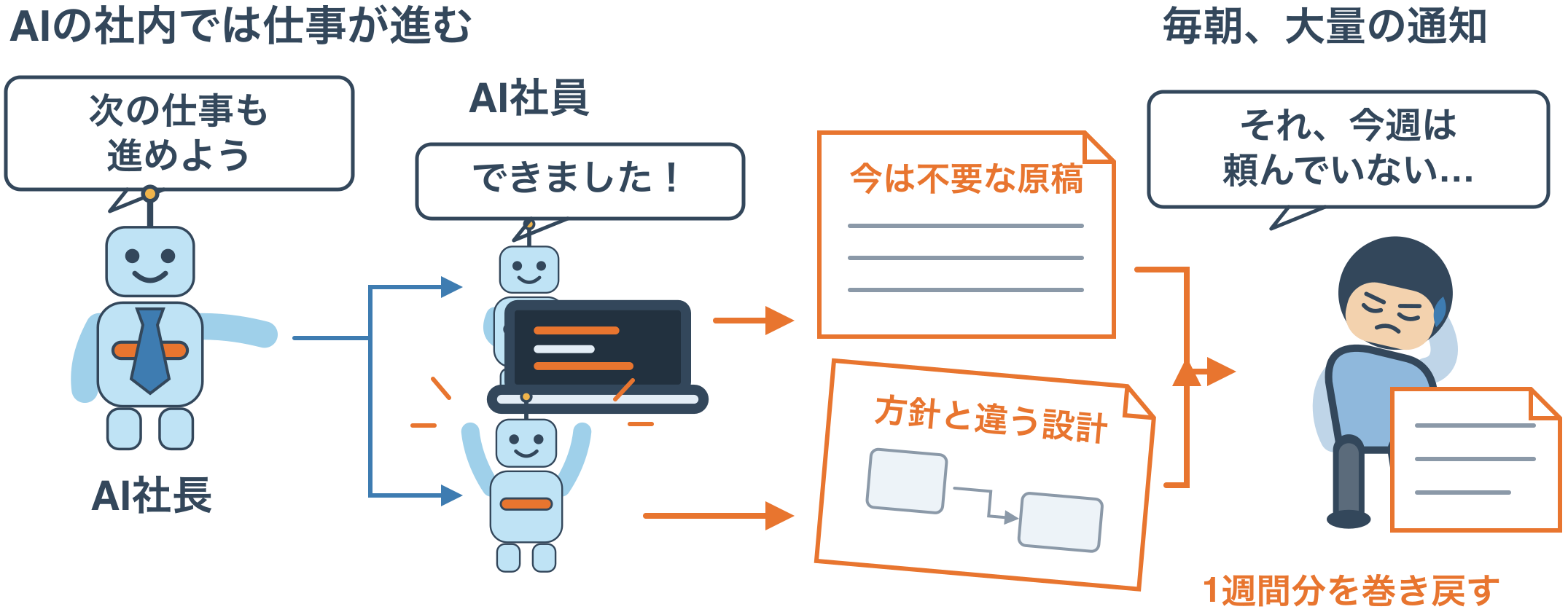
人は社長・会長役。目標と予算を渡し、成果と費用で判断



余談：AI会社を畳んだ理由

出典：Alien Brain Trust「Why We Killed Paperclip」（2026.4）

自分で作るより、点検と手直しに時間がかかった



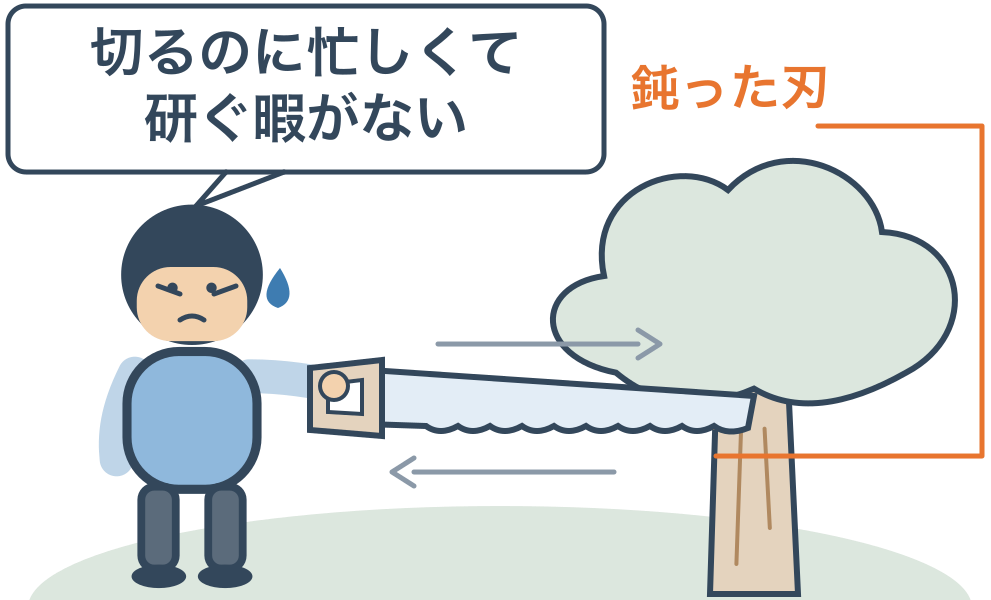
停止後 → 人が仕事を一つ選ぶ → AIは実行・報告して、止まる

木こりのジレンマ

出典：FranklinCovey「Sharpen the Saw」／AIへの応用：講師の考察

目先の作業に追われ、効率を上げる準備ができない

元のたとえ

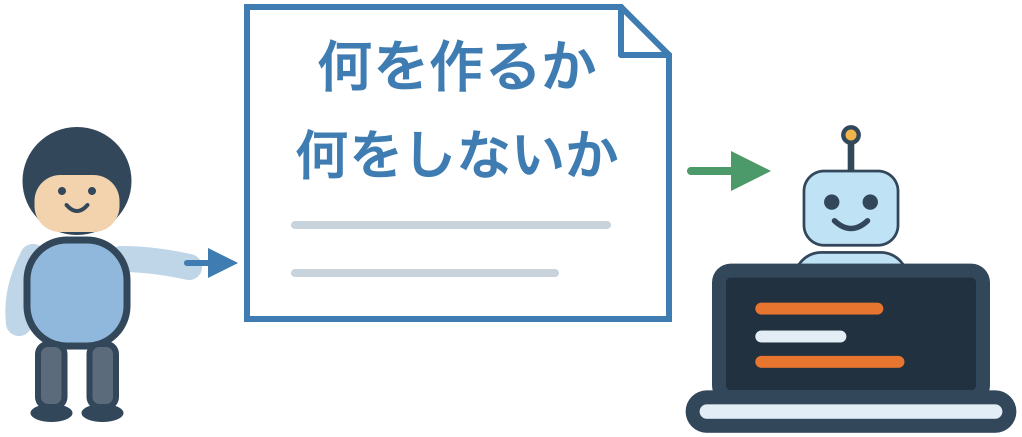


鈍い刃で続けても、仕事が進まない

一度止めて研ぐ → 後の作業を速くする

AIコーディングでは

研ぐ＝計画・分担・評価を整える



意図を伝え、プランで確認

プランモードで足りれば、そこまで

実装へ

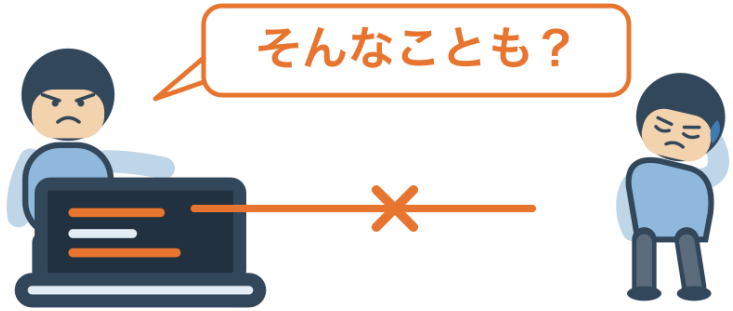
切る木を選び、必要な準備が済んだら始める

ブルックスの法則への答え

Netflix企業文化資料　　AIへの適用と結論：講師の考察

AI モデルをたくさん雇うとブルックスの法則が再発しちゃう
⇨ 少数精鋭の AI モデルに，音声でたくさん話しかけてプラン作り

Netflixの採用

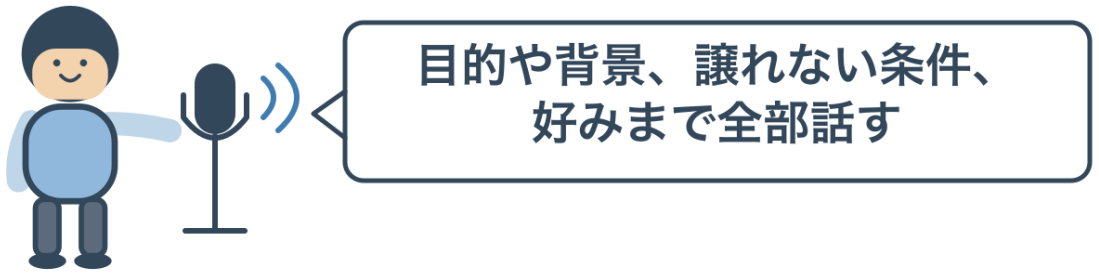


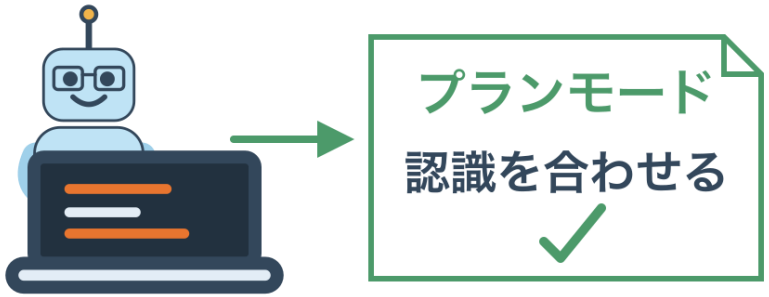
ブリリアントジャークを採らない

AI時代の「採用」

混乱を増やす低品質なモデルは使わない

選んだ賢いAIに、音声で話し尽くす





おまけ：さらに学びたい方へ

Anthropic：公式ガイド　Matt Pocock：grill-me　堤 大貴：自己改善の設計

プランの練り方から、自己改善の設計まで

プランを練る

Claude Code 公式ガイド

調べる → 計画 → 実装

AIに質問してもらう依頼例
プランを省いてよい仕事

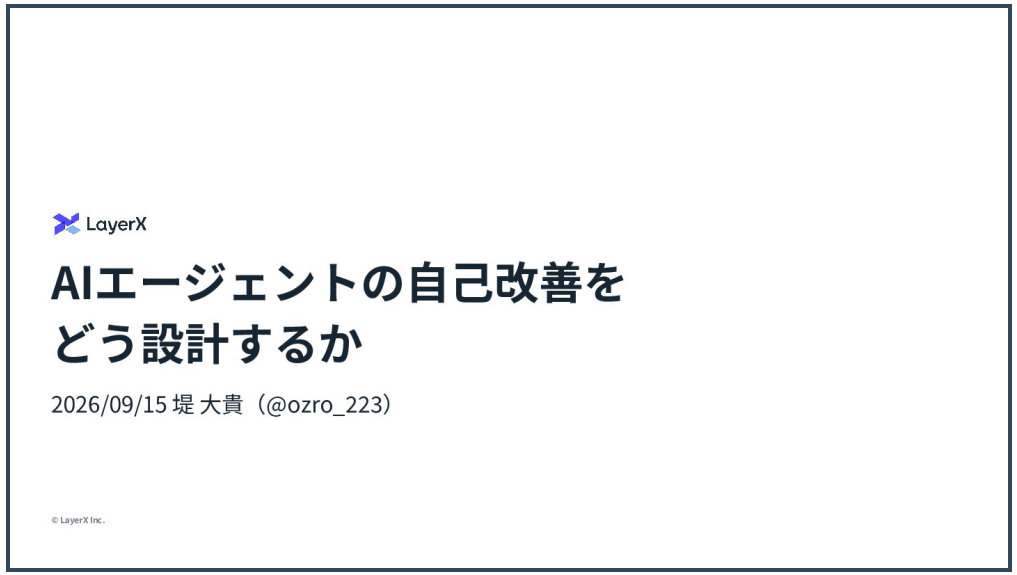
grill-me

Matt Pocock 公開スキル

回答に応じて、次の質問へ

曖昧な条件を一緒に詰める

改善の仕組みを作る



堤 大貴 氏（LayerX）

停滞する理由：7〜10頁
設計と評価：17〜20頁

おまけ：ブリリアントジャークの逆は？

出典：DeMarco・Lister『Peopleware』第2章

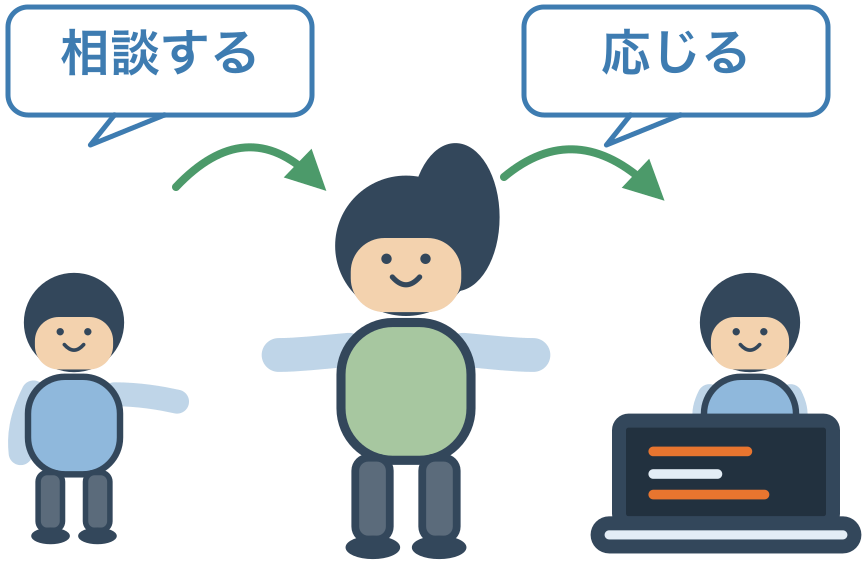
本人の腕前だけでは見えなかった、チームへの貢献

上司から見た彼女

開発もテストも
特に優秀ではない

本人の開発・テスト能力

チームでの彼女



会話と協力が生まれる

在籍
12年間

参加した案件は
すべて成功
✓

チームの「触媒役」

それでも、上司には価値が伝わらなかった。

おまけ：Obsidianでデジタルツイン

Digital Brain　obsidian-skills　人物像：講師のObsidian記録（2026.09）

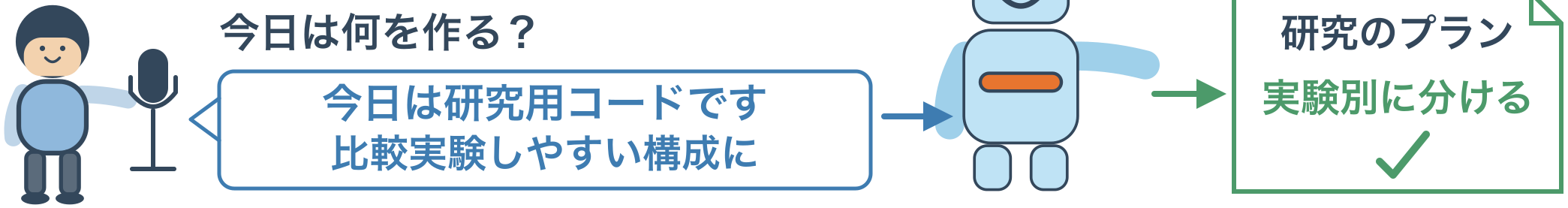
研究と趣味。同じ自分でも、作る目的は違う

Digital Brain
記録をAI用の記憶へ整理する

obsidian-skills
AIからノートを読み書きする

AIがまとめた僕：Obsidianの記録

判断　サンクコストは重みゼロ
資料　意味のない配色を嫌う
趣味　「単一ファイル主義」

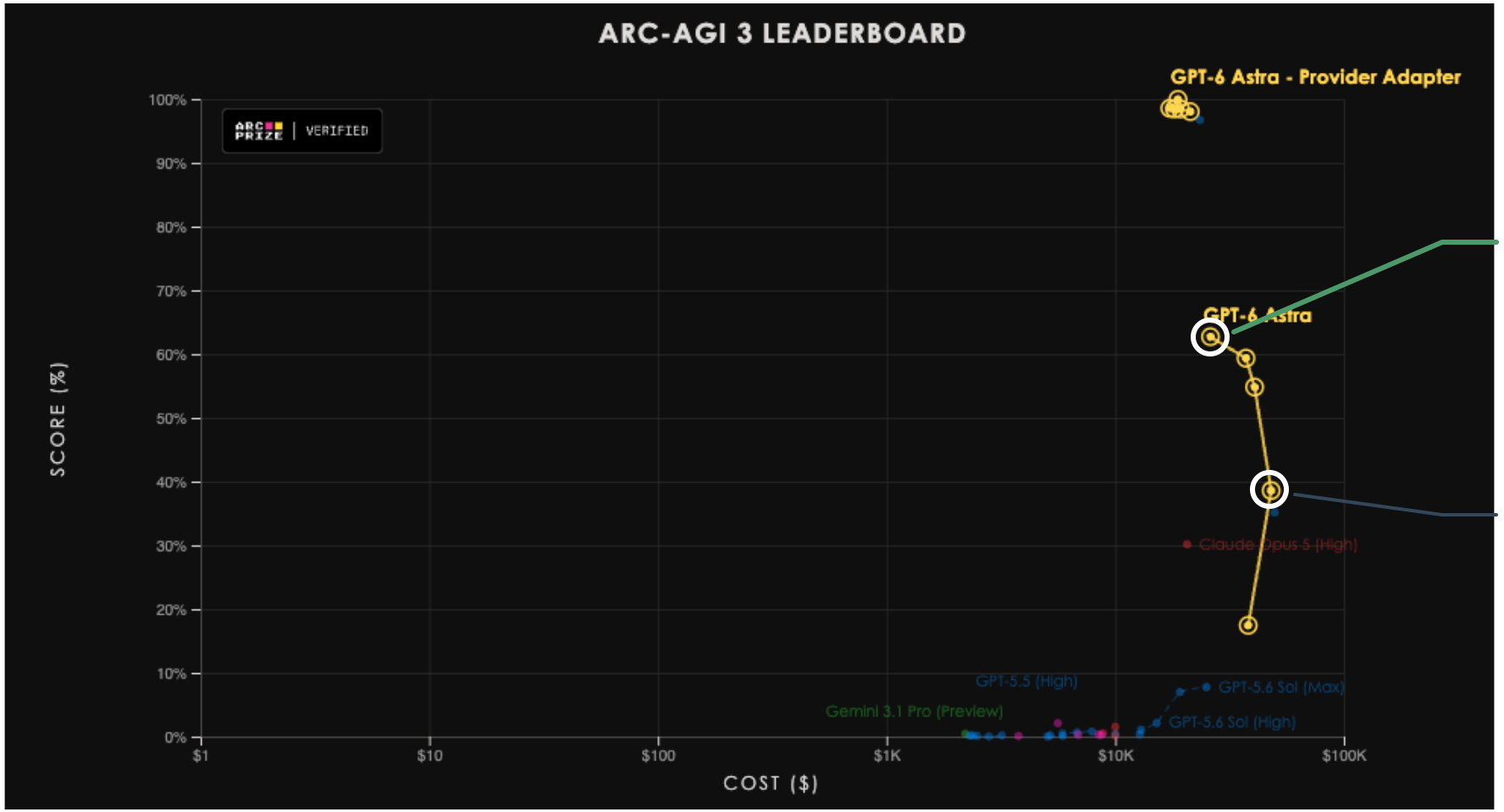


分身を育てるまめさがあるなら、プランもきっと練れる

おまけ：考え抜いた方が安い？

出典：ARC Prize「OpenAI's GPT-6 Astra on ARC-AGI-3」（2026.9）

Astra では、エフォート Max の方が寄り道なしで安くなることも



Standardで比較

Max
62.7%
$26,098

Medium
38.6%
$48,090

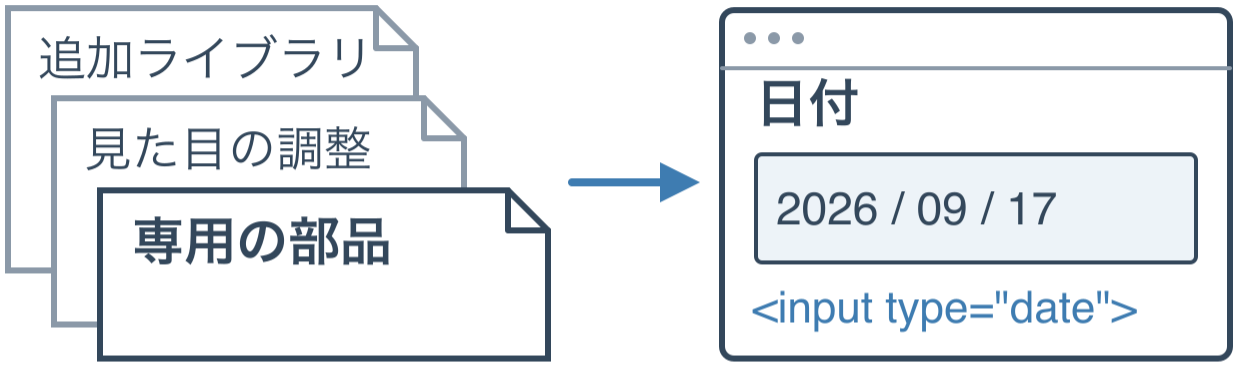
呼出し回数と
総トークンが減る

おまけ：ponytail

出典：DietrichGebert「ponytail」スキル

必要な機能と安全性を守り、AIの作りすぎを抑える

例：日付の入力欄がほしい



新しい部品を積み上げる　　**標準の入力欄で足りる**

作る前に、この順で確かめる

1. 本当に必要か
2. 既存コード・標準機能で足りるか
3. 導入済みの道具を使えるか
4. **足りない分だけ実装する**

3段階のモード

lite　代案を添える

依頼どおりに作り、代案も示す

full（標準）　簡潔に作る

既存機能を優先して実装する

ultra　要求から疑う

不要なら、作らない提案も

おまけ：sanitize-artifacts / natural-japanese

出典：手元のsanitize-artifacts定義　　coji「natural-japanese」

制作の事情を消し、読者に伝わる日本語に直す

sanitize-artifacts

読者に不要な制作事情を取り除く

~~ご指摘を反映しました。~~
~~先ほどの案を修正しました。~~

保存ボタンを押すと、
処理が始まります。

削る：修正への返事・却下案・編集事情
残す：読者に必要な事実・条件・出典

依頼例：制作中のやり取りを消して、
そのまま配れる完成稿にして

natural-japanese

構成から文の癖まで整える

保存ボタンを押下することにより、
処理が開始されるという点が重要です。

保存ボタンを押すと、
処理が始まります。

構成を決めて書き、定型句や直訳調を検出
一括置換せず、前後の意味を見て直す

write：書く・直す
score：文書は変えず、問題点を診断する