# AI駆動開発の実践

自社サービスと個人開発での活用事例

2025.12.03@インターシステムズジャパン株式会社

Sumihiro Kagawa

BRAVETECHNOLOGY

# 自己紹介

# 加川 澄廣（かがわ すみひろ）

## ❖略歴

- ✦LINE API Expert (2019.04〜)
- ✦株式会社ブレイブテクノロジー 取締役（2021.04〜）
  - LINE Developers Community での活動がきっかけで現職へ
  - 様々な LINE ミニアプリを開発中

## ❖趣味

- 個人開発
- e-bike でゆるゆる走る
  - 時にはアワイチ、カスイチなど 100km 超えの長距離も走ります

sumihiro3
sumihiro.kagawa
s.kagawa@bravetechnology.co.jp

# BRAVE TECHNOLOGY


# MAKE A HAPPY

お客様が利用するときの操作性を重視し、
迷わずに使えるユーザーインターフェースをつくる。
当社が長年取り組んできたアプリケーション開発のノウハウを活かしサービスを提供しています。

https://bravetechnology.co.jp/

BRAVE TECHNOLOGY

## 認定パートナー


株式会社ブレイブテクノロジーは、LINEヤフー株式会社のパートナー制度に認定項目として新設された「Technology Partner」の「LINEミニアプリ部門」において、初回のパートナーに認定されました。「LINEミニアプリ部門」では、提供する「LINEミニアプリ」のサービス導入実績と、アクティブに活用されている店舗数を基準に「LINEミニアプリ」事業の拡大に貢献したパートナーが認定されています。

LINE Technology Partner 詳細はこちら 》

LINEで順番待ち

# matoca

マトカ

気軽に待ってもらえる順番管理サービス

**3密の対策**

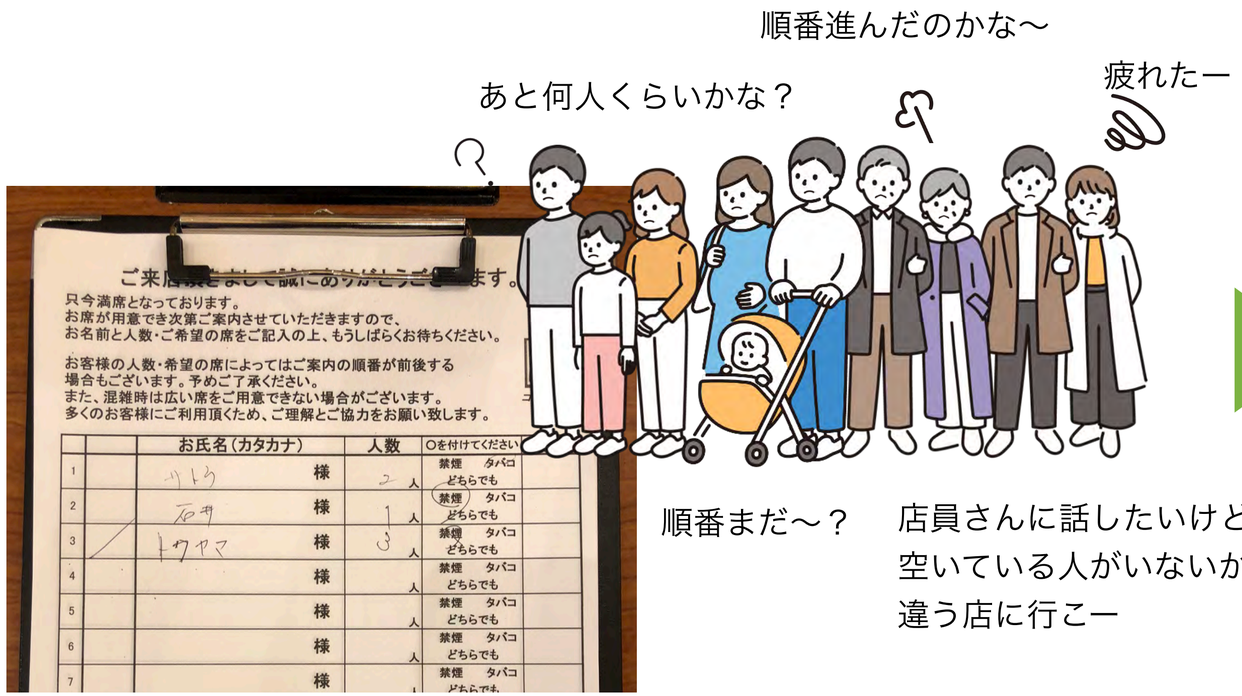
密集、密室、密接
を避けるサービス

Social
Distance

BRAVETECHNOLOGY

# 『順番待ちを手軽に管理できる整理券システム』

## 順番待ちの混雑




店員さんに話したいけど
空いている人がいないから
違う店に行こー



**機会損失**や**お客様満足度の低下**

など様々な課題

タブレットやLINEミニアプリから整理券を発券することで

順番が近づくまで**並んで待つ必要がなくなります**

# 導入企業・店舗

matoca(マトカ)はさまざまな業種に対応し、導入企業が続々と増えています。

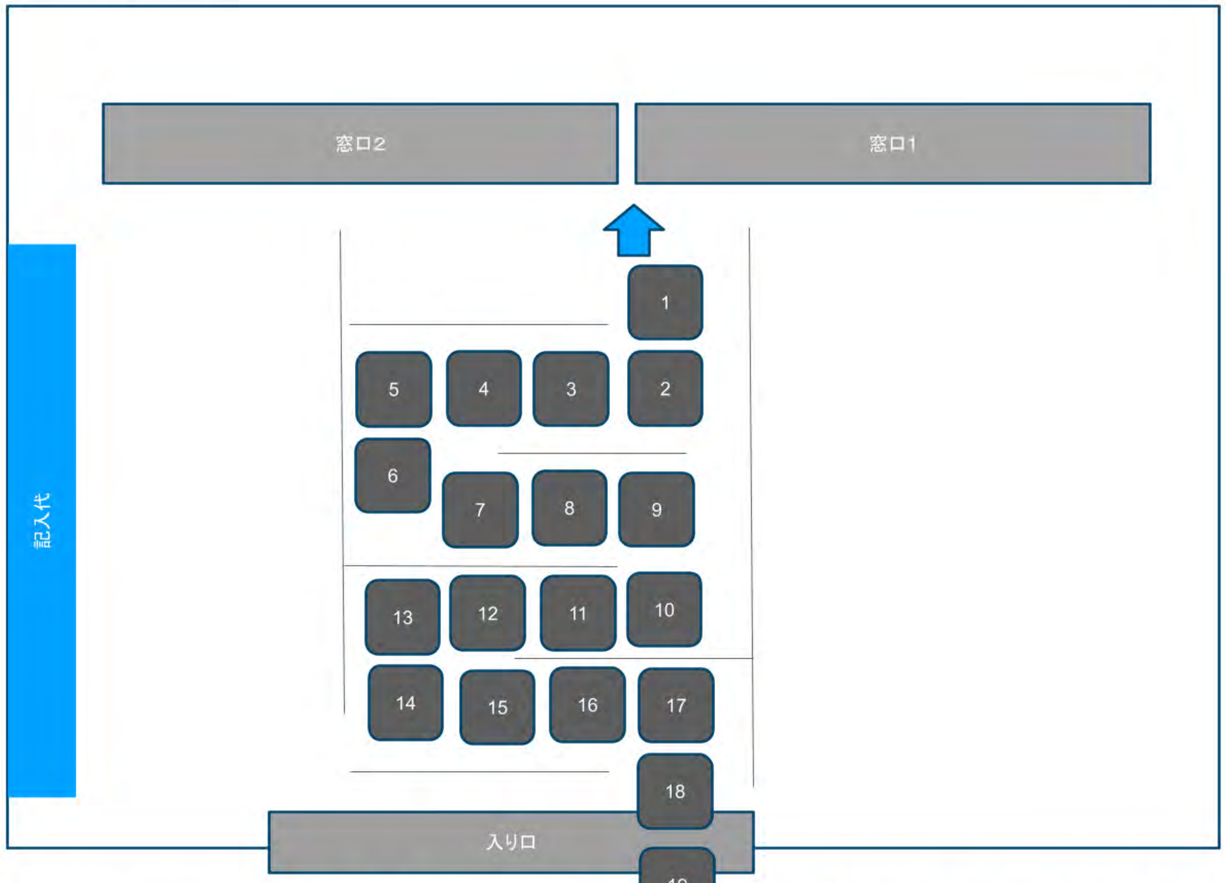
導入企業 **440** 社超　　導入店舗 **3000**店突破!!

| | |
|---|---|
| 飲食 | 珈琲所コメダ珈琲店 / 一極上すしの美登利 / 8番らーめん / つばめパン / 一風堂 / お好み焼みっちゃん総本店 / すし遊館 / 大戸屋ごはん処 / 焼肉叙々苑 / うな勢 / 鮮うまい鮨勘 / 株式会社海元 Kaigen / 釜炊きごはんとハンバーグ ワイナ食堂 / 明太子をつくってよかった。博多中洲ふくや / 博多の味やまや / STEAK 88 HOUSE |
| 医院・クリニック<br>調剤薬局 | お倉が浜Kidsクリニック / 医療法人上野会 上野会クリニック / いまにし胃腸肛門科 / とりい皮膚科クリニック / うつべ動物病院 / CLOVER / 岩槻 いまい眼科 / 足と歩行の診療所 Foot&Walk clinic / すずな皮ふ科クリニック / 江戸堀サンテクリニック / かつれん耳鼻科クリニック |
| 商業施設 | TRIAL / OIOI / SEIBU SOGO / TOKYO MYSTERY CIRCUS / サンエー / ハウステンボス / 大丸福岡天神店 |
| 小売 | UP GARAGE / らしんばん / e☆イヤホン / 駿河屋 / DEAN & DELUCA / TSUBURAYA / koala |
| 理美容 | PLAGE BARBER & BEAUTY / 美容室 HAIR'S GATE / kamuy hair / DANK / NAMIchOSU Hair Salon 髪ちょす / カットひまわり / Barber 理容ドラゴン DRAGON |
| 自治体・その他 | 吹田市 Suita City / 尼崎市 Amagasaki City / 別府市 / 多可町 Taka Town / 西宮市 NISHINOMIYA CITY / 宮崎県 都城市 / 永平寺町 EIHEIJI / 天川村 / サウナ道場 / Yokohama F·Marinos / SANFRECCE HIROSHIMA / JR西日本 / カラオケ歌屋 / 東京都北区立 元気ぷらざ |

導入企業一例：2025年2月現在

# みどりの窓口導入事例

## 導入前

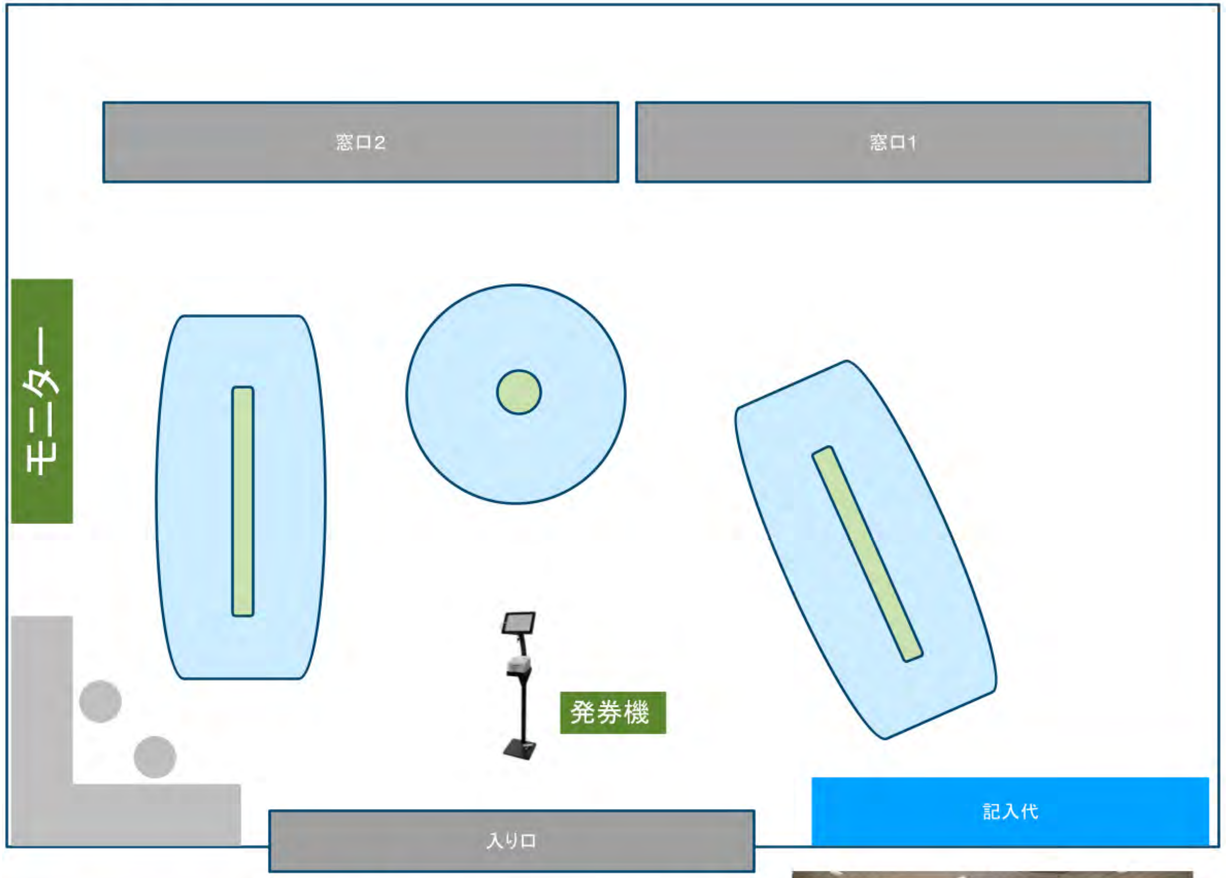



## 導入前は長蛇の列ができていた@JR 大阪駅

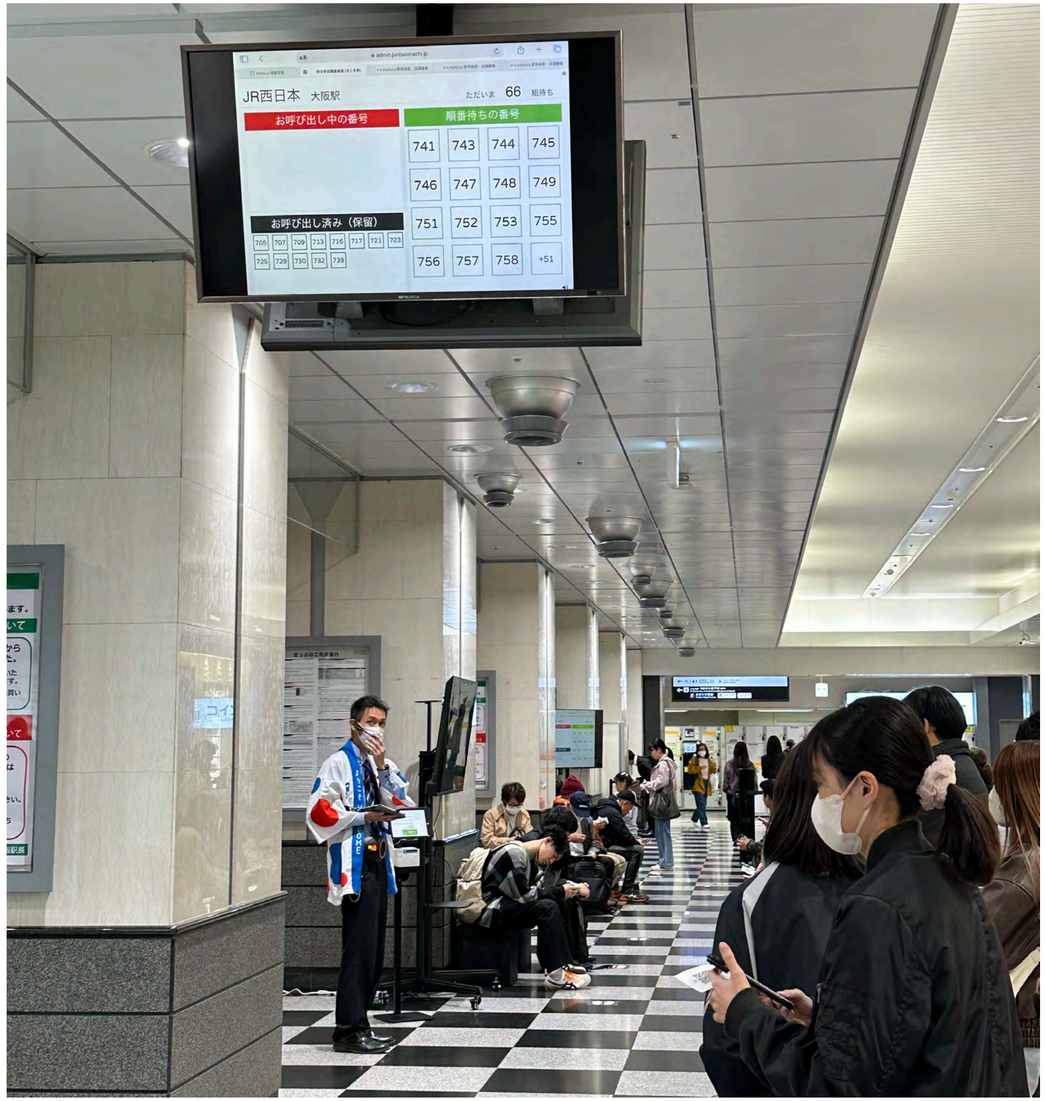

BRAVETECHNOLOGY

# みどりの窓口導入事例

導入後




**導入後は長蛇の列は無くなり、椅子に座って待ったり、場を離れて用事を済ませたりできている@JR 大阪駅**




BRAVETECHNOLOGY

00

# アジェンダ

# AI駆動開発の特徴から、自社サービス・個人開発での具体的な実践事例を通して、その有効性をご紹介します

1.AI駆動開発とは

2.自社サービスでの活用事例

3.個人開発での活用事例

4.まとめ

# 本題に入る前に
# 軽く生成AI関連の話題

# 現在利用しているAIツール

Claude

Gemini

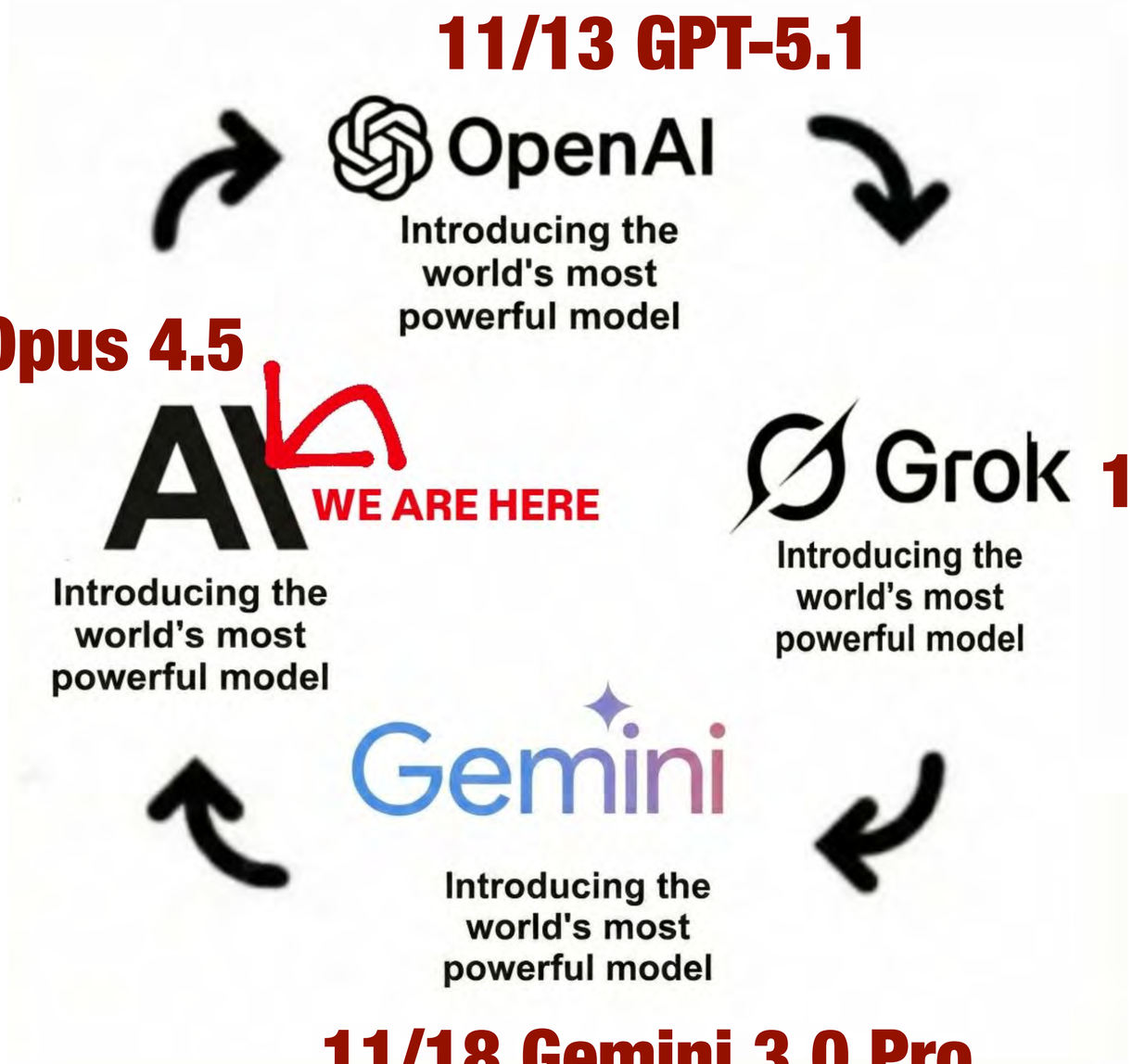
GitHub Copilot

CURSOR

Google Antigravity

# 最近の生成AI界隈

11/13 GPT-5.1




11/25 Claude Opus 4.5

11/17 Grok 4.1

11/18 Gemini 3.0 Pro

https://x.com/marswalkerr/status/1993031054052278566?s=46&t=3bwMDrHEznfeD16hYHUCvw

01

# Part1. AI駆動開発とは

# AI駆動開発（AI-Driven Development）とは

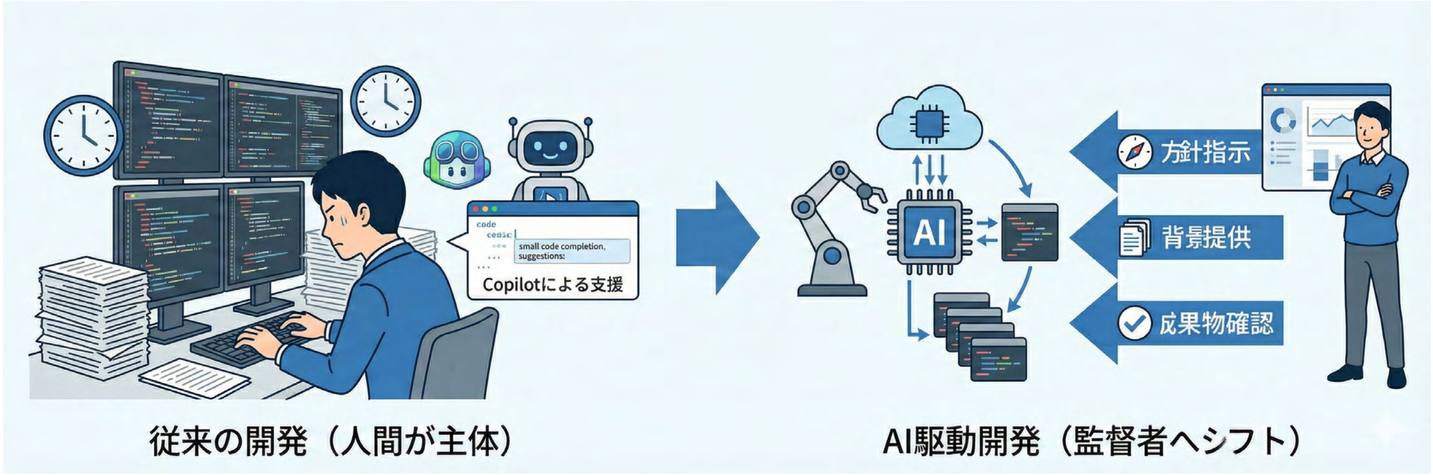
AIツールの導入ではなく、**「開発の主体」**と**「人間の役割」**の再定義である

## ❖定義

- 全ての開発プロセスにおいて**AIを主体**とし、人間側は**「方針決定」** **「背景情報の提供」** **「レビュ」**の3点のみに専念する開発アプローチ

## ❖パラダイムシフト

- これまで：人間が手を動かし、AIが補助する（Copilot）
- AI駆動開発：AIが手を動かし、人間が監督する（Autopilot / Agent）



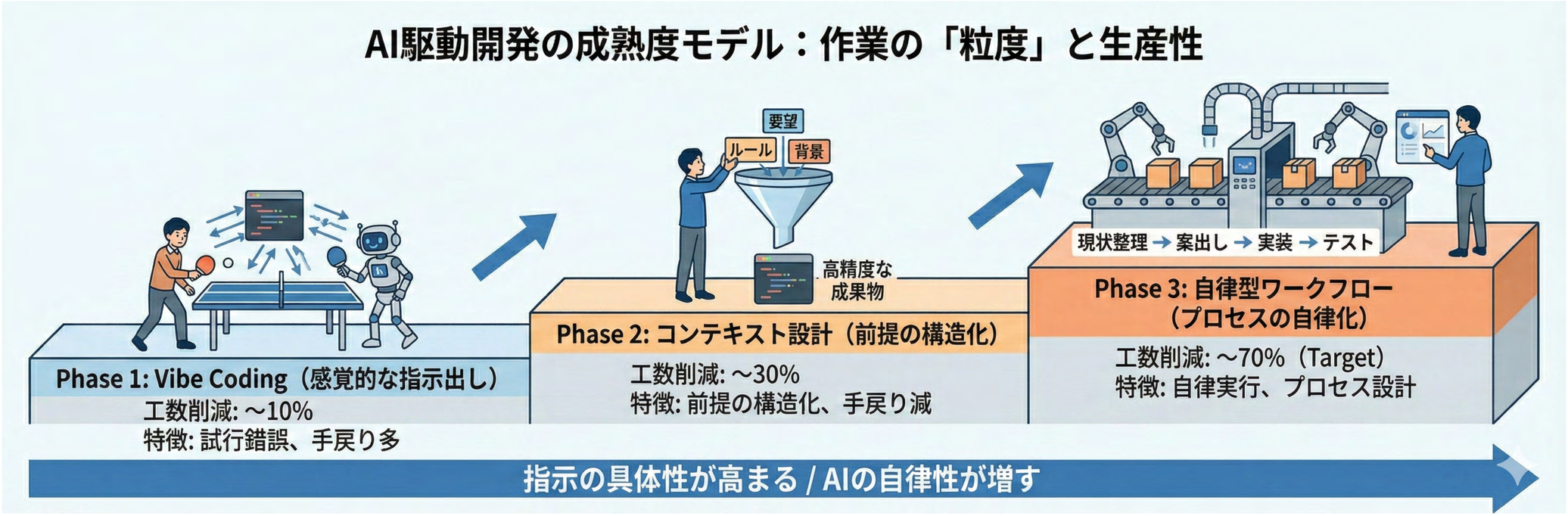
# AI駆動開発の成熟度モデル：作業の「粒度」が生産性を決める

人間がいかに精密に「作業設計」を行えるかで、工数削減効果が大きく変化する

## ❖AI駆動開発の成熟度モデル

### ✦AI駆動開発は、指示の具体性とプロセスの設計深度によって3つの段階に分類される

- 段階が上がるほどAIの自律性が高まって手戻りが減少し、削減効果が最大化する



# Phase 1: Vibe Coding（感覚的な指示出し）

圧倒的な初速が出るが手戻りの多さで相殺されることも多い


## ❖概要

- 「これ作って」と要望だけを投げ、出力を見て修正するスタイル
- 品質判断や前提整理は行わず、AIのスピードに頼る

## ❖メリット

- コード試作が秒単位で完了。「まず動くもの」を作るプロトタイピングに最適

## ❖効果など

- 「コードを書く時間」は激減するが、コンテキスト不足により意図とズレた出力が出やすい
- 結果「修正のラリー」に時間を取られ、トータルの生産性向上は限定的となりがち
- **再現性が低い**
- ただし、**最近のAIモデルやツールでは「分析→意図の整理→タスクリスト化」することができてきた**

# Phase 2: コンテキスト設計（前提の構造化）

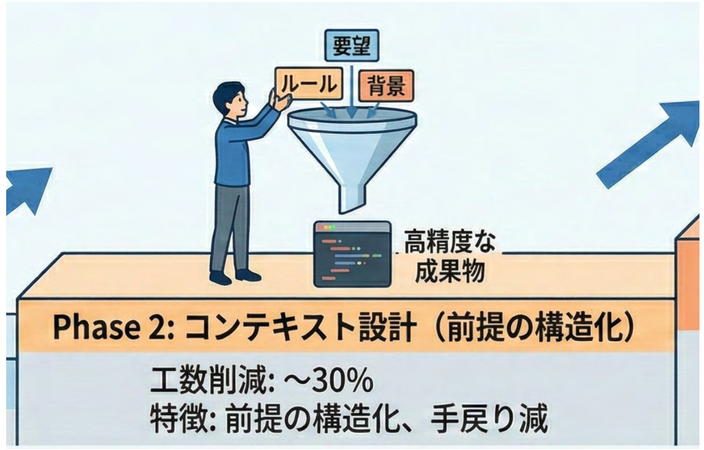
AIが「迷わない」環境を作ることで、手戻りを未然に防ぐ

## ❖概要

- 要望だけでなく、「**前提情報・制約・評価基準**」をセットで設計して渡す段階
  - 例：DBスキーマ、Open API定義、Unit Test、デザインガイドライン、既存コードの設計思想など

## ❖効果など

- AIに「正解の基準」を与えてから走らせるため、出力精度が劇的に向上する
  - AIに「正解」を考えさせるのではなく、**AIに「正解に合わせてもらう」**
- 「指示→ズレ→修正」の往復回数が減り、実用的な工数削減につながる



# Phase 3: 自律型ワークフロー（プロセスの自律化）

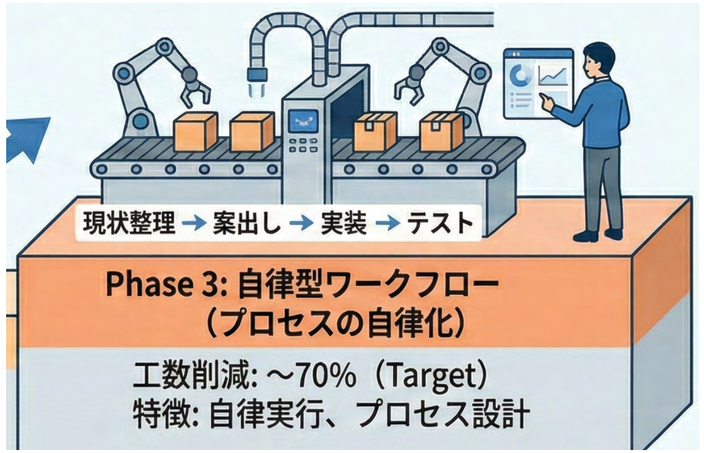
「点」のタスク依頼から、「線」のプロセス委任へ

## ❖概要

- 「現状整理→案出し→実装→テスト」という**ワークフロー**を定義し、AIに自律実行させる
- 人間は途中の微調整から解放され、プロセス設計と最終確認に集中する

## ❖効果など

- ここまで来ると、人間は実行者から「監督者」へと完全にシフトする
- **プロセス全体の再現性が高まる**
  - 要件変更による手直しから、短期間での再構築へ



# AI時代のエンジニアの責任：「書く力」から「見る力」へ

実装スピードが上がるほど、「ビジネス価値」と「整合性」のレビューが生命線になる

## ❖役割のシフト（Shift in Responsibility）

### ✦Coding (Before)

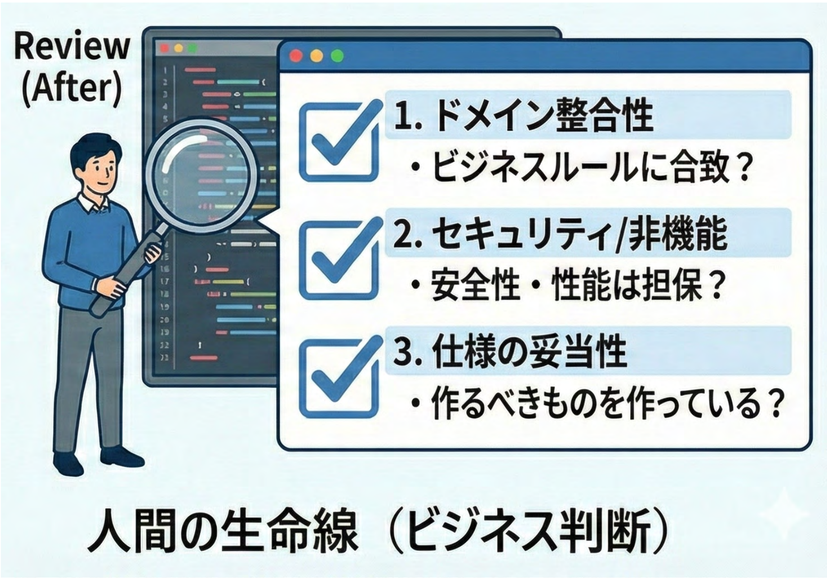
- 構文エラーがないか、動くか。（AIが代替可能）

### ✦Review (After)

- **ドメイン整合性**：ビジネスルールや顧客の要件に合致しているか？
- **セキュリティ/非機能**：パフォーマンスや安全性は担保されているか
- **仕様の妥当性**：そもそも「作るべきもの」を作っているか？

## ❖結論

- AIは「正しく動くコード」は書けるが、「正しいビジネス判断」はできない。そこが人間の砦となる



# 属人化からの脱却：SDD（仕様駆動開発）への回帰

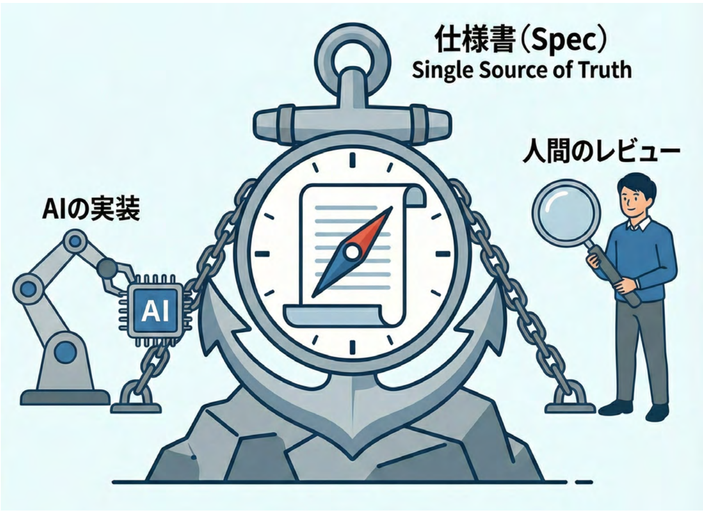
明確な「仕様書」こそが、AIに対する最強のコンテキストである

## ❖課題

- 「Vibe Coding」や「コンテキスト設計」は、個人のスキルに依存しやすい→**再現性が低い**

## ❖解決策の一つとして「仕様駆動開発 (Specification Driven Development)」

- Markdown等で記述された「**仕様書（Spec）**」を正（**Single Source of Truth**）とする
  - 職人芸ではなく、AIに渡せる形式にしておく
- 人間が仕様を書き、AIが実装し、人間が仕様に基づいてレビューする
- 統合テストなども仕様書からできるようになる（かも）



# AWS が策定するAI主導開発ライフサイクル（AI-DLC）

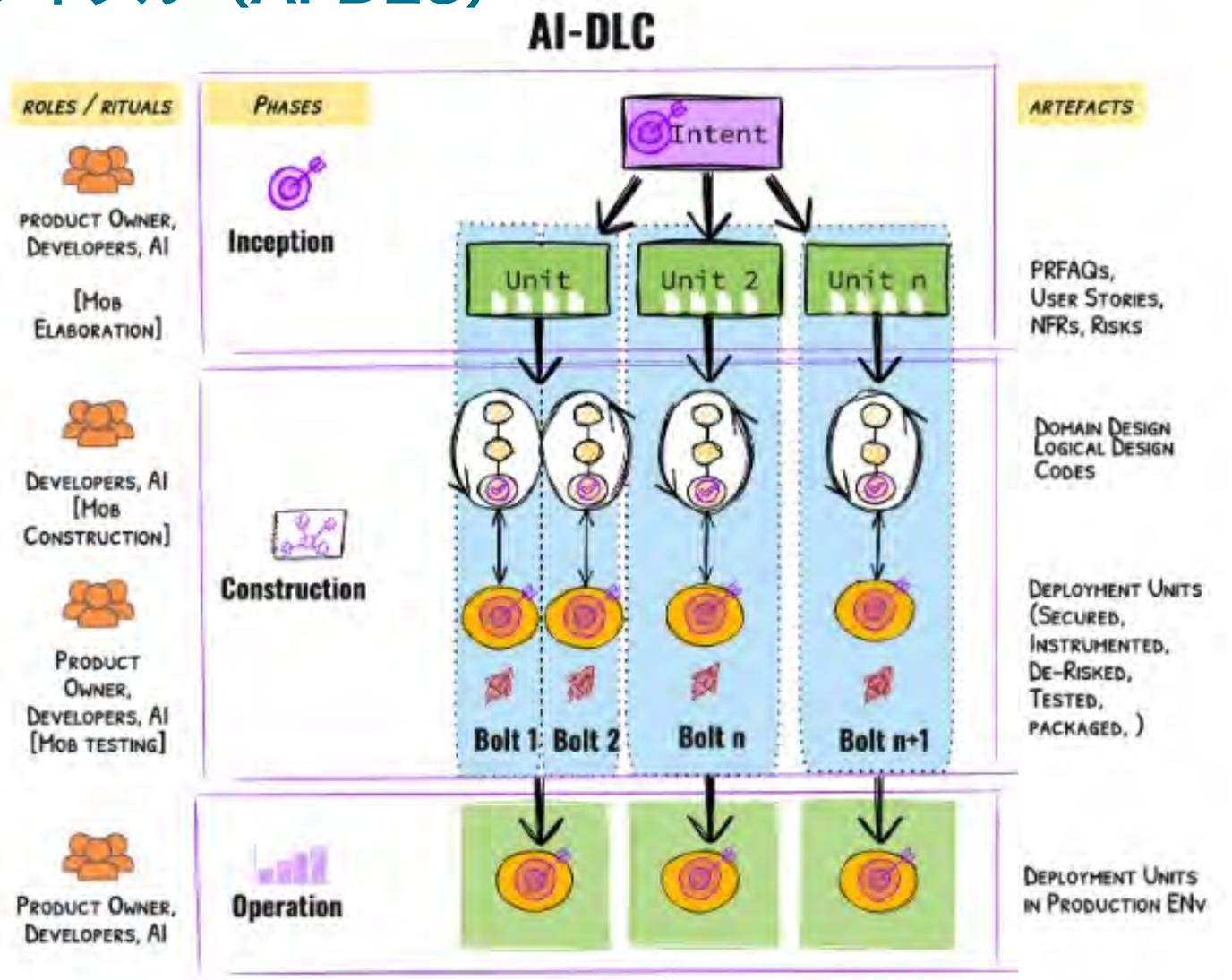
参考：AWS の AI-Driven Development Lifecycle (AI-DLC) Method Definition より

## ❖AI-DLCとは

- AWSが策定しているAIの速度とエージェント機能を最大限に活用するために、開発プロセスを根本的に再構築することを目的にした開発プロセス
  - Kiro はAI-DLCに沿った開発を支援

## ✦AI-DLC の原則

- AIがタスク分解や計画立案を主導し、**人間は重要な意思決定ポイントにおいて承認、検証、および監督**の役割を担う
- 数時間または数日単位の「**ボルト**」という迅速なサイクルを回して開発を進める




https://prod.d13rzhkk8cj2z0.amplifyapp.com/

# 次のトレンド：SDD（仕様駆動開発）へ

参考：AWS の AI-Driven Development Lifecycle (AI-DLC) Method Definition より

## ❖AI-DLC での原則の一つ「9. MINIMISE STAGES, MAXIMISE FLOW（段階を最小化し、フローを最大化する）」より

- *Through automation and convergence of responsibilities, AI-DLC aims to minimize the handoffs and transitions, enabling continuous iterative flow. But human validation and decision-making remain critical to ensure that AI-generated code does not become rigid ('quick-cement') but stays adaptable for future iterations. To address this, AI-DLC incorporates minimal but sufficient number of phases specifically designed for human oversight at critical decision junctures. These validations act as a form of 'loss function', by identifying and pruning wasteful downstream efforts before they occur.*
- AI-DLCは、自動化と責任の統合を通じて、引き継ぎや移行を最小化し、継続的な反復フローを実現することを目指す。しかし、AI生成コードが硬直化（「クイックセメント」）せず、**将来の反復に適応し続けられるよう、人間の検証と意思決定は依然として重要**である。この課題に対処するため、AI-DLCは重要な意思決定ポイントで人間の監視を可能にする最小限かつ十分な数のフェーズを組み込む。これらの検証は「損失関数」として機能し、無駄な下流作業が発生する前に特定・剪定する。

https://prod.d13rzhkk8cj2z0.amplifyapp.com/

# SDD（仕様駆動開発）への取り組み

プロンプトを書く時代から、「**仕様（Spec）**」を書く時代へ

## ❖AI開発ツールはこの「構造化」のアプローチを製品に取り込み始めている

### ✦SDD実現を目指しているツール

| 名称（開発元） | 特徴 |
| --- | --- |
| Kiro（AWS）  | AWSのAI-DLCに準拠したIDEでGUIベースでSDDを実施できる<br>https://kiro.dev/ |
| Spec Kit（GitHub）  | フェイズを細かく分けて、前フェイズまでに作成したドキュメントを参照しながら、AIが自立して実装を進められるようドキュメントをきっちり整備する<br>⭐ 52.1k https://github.com/github/spec-kit |
| cc-sdd（Gota）<br>※日本人エンジニア | Kiro と同じフローで多くのAIコーディングエージェントで利用できる。日本語対応。軽量で中規模以上の機能追加向き<br>⭐1.9k https://github.com/gotalab/cc-sdd |

# AI駆動の思考は、そのまま「プロダクトの品質」になる

AIを制御するために学ぶ「コンテキスト設計」や「ワークフロー」は、ユーザーに提供する**AI機能の裏側そのもの**である

| 開発プロセス（Internal） | プロダクト機能（External） |
| --- | --- |
| ✦AIにコードを書かせる時、ただ「書いて」と言っても良いコードは出ない<br>✦解決策：コンテキスト設計、ワークフロー構築 | ✦例：GitHubリポジトリ解説動画生成ツール<br>❌ NG（Vibe Coding的）<br>▸ 「このリポジトリの動画台本を作って」と投げる → 浅い台本<br>✅ OK（Agentic Workflow的）<br>▸ READMEから「解決課題」を抽出<br>▸ コード構造から「技術スタック」を分析<br>▸ 構成案を作成（導入→スター数推移→技術解説）<br>▸ 台本執筆 |

02

# Part2. 自社開発での活用事例

# Case1: 決済連携ミドルウェア開発での活用

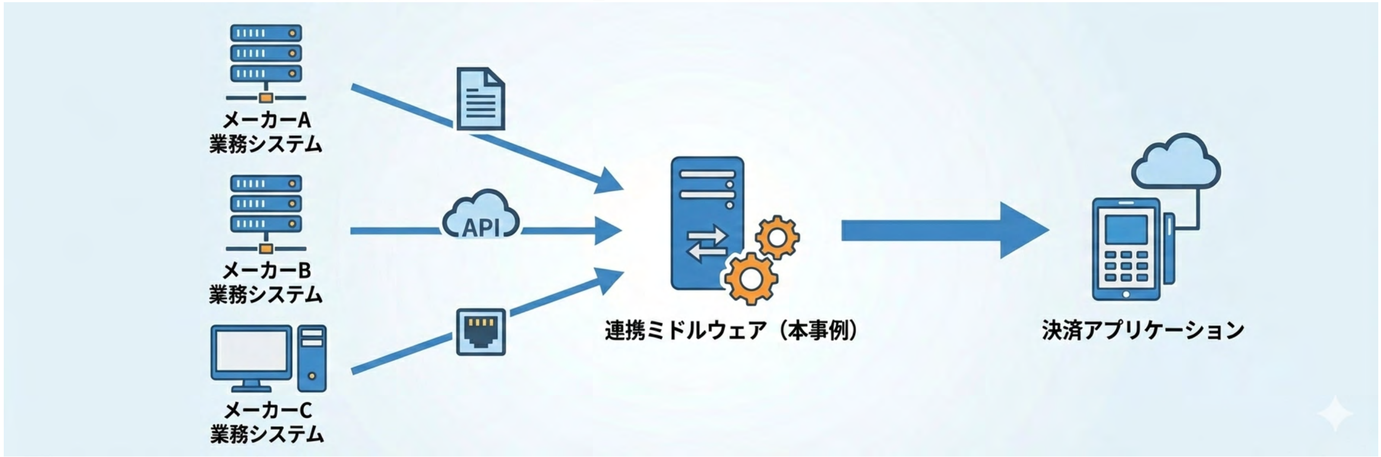
多様な業務システムと決済端末をつなぐ「変換」のハブ

## ❖プロジェクト概要

- 導入先ごとに異なる「業務システム」と、「決済アプリケーション」の間で、データ連携・変換を行う仲介アプリ
  - **主な機能**：請求情報、決済履歴の相互変換や、それら情報の管理（DB）

## ✦課題

- メーカーごとに連携仕様がバラバラ（CSV、Web API、TCPなど）
- 導入先が増えるたびに、膨大な「変換ロジック」の実装とテストが発生する



# 実践：仕様書 × 内部設計 = 変換コード生成

現在のステージ: Phase 2: コンテキスト設計（個人活用レベル）

## ❖AIへの入力（Context）

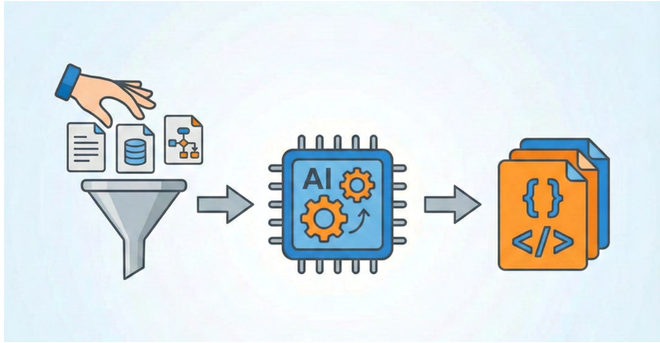
- **外部仕様**：連携先システムの仕様を、AIが理解しやすい形式（Markdown等）に構造化したもの
  - **MarkItDown** などを利用して、AIが理解しづらい形式（PDF, Word, Excel など）から変換
- **内部仕様**: 自社アプリのデータ構造（エンティティ定義）のドキュメント

## ❖AIの出力

- ファイル連携での変換処理用定義ファイル（yaml形式）
- API連携での変換処理クラス（TypeScript）

## ❖成果

- 「仕様調整（コード値の変換など）」の手動修正は残るものの、
  定型的な変換コードの実装工数は大幅削減




https://github.com/microsoft/markitdown

# 応用：ファイル連携を元にしたソケット通信処理の実装

設計思想を統一した実装の生成

## ❖課題

- 既存の「CSV連携」に加え、難易度の高い「TCPソケット通信（固定長・繰り返し項目あり）」の追加要件が発生

## ❖AIへの指示

- ❌「固定長電文のパース処理を実装して」
- ✅「既存のファイル変換処理と同様のインターフェース・構成（項目の変換設定ファイル構造など）を踏襲して実装して」と指示（具体的なクラスなどを指定）

## ❖結果

- プロトコル（CSV vs TCP）は全く異なるが、**コードの設計思想が統一された実装**が生成できた

# 学びと課題：個人最適から全体最適へ

## ❖学び（What vs How）

- 「何を作るか（What：連携仕様）」だけでなく、**「どの設計パターンで作るか（How：既存のインターフェース）」**をコンテキストとして渡すことで、保守性の高いコードが生成される

## ❖今後の課題

- **細かい仕様の不整合**：コード値変換（例: '01' → ‘Cash'）などの微調整は、まだ人間が介入している（自動化しきれていない）
- **属人化**：現在は特定の個人がこの「プロンプト設計」を行っている状態

## ❖Next Step

- この変換ルールや設計思想を「仕様書（SDD）」として標準化し、チーム全員が同じ品質でAIを使えるようにする

03

# Part3. 個人開発での活用事例

# Case 2: 完全自動動画生成エージェント「Yukkuri Video Studio」

1コマンドで人気 GitHub リポジトリの解説動画を生成する

## ❖What

- 日付（動画配信日）を渡すだけで、リポジトリの選定・分析・台本・音声・動画レンダリングまで行う自律型エージェント
  - 動画レンダリングはプレビュ後に別途実施している

## ❖Why

- 毎日の動画投稿を「ワンオペ」で回すため
- 人間はレンダリング前のプレビュで最終チェックと多少の手直しに注力する
  - 手直しは、正しく発音させるための用語登録や、言い回しの調整程度

## ❖Tech Stack

- **Brain**：Claude 4.5 Sonnet + Gemini 2.5 Flash
- **Renderer**：Remotion (Reactベースの動画生成)
- **Voice**：AquesTalk 10

# Case 2: 完全自動動画生成エージェント「Yukkuri Video Studio」

1コマンドで人気 GitHub リポジトリの解説動画を生成する


https://protopedia.net/prototype/7675

# 「台本を作って」とは頼まない

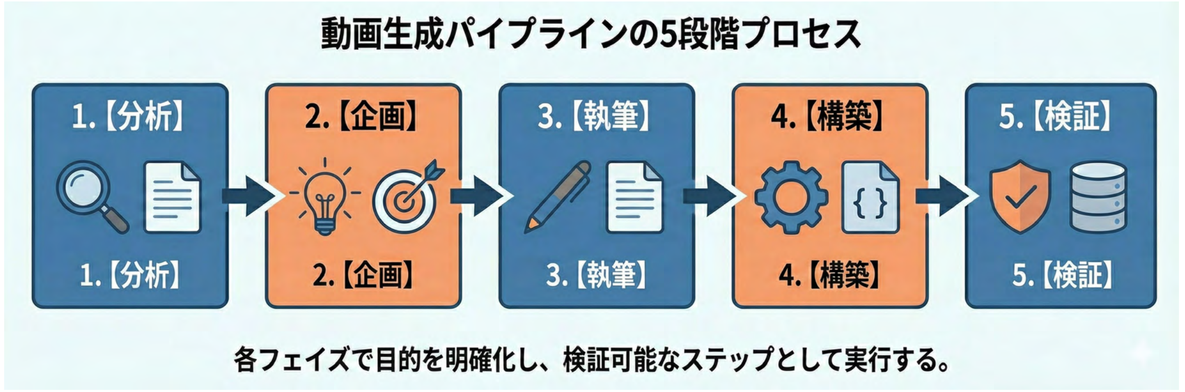
単に「動画の台本を作って」とは頼まない。工程をパイプライン化してAIに道筋を示す

## ❖❌ NGパターン

- 「このリポジトリの解説動画の台本を作って」
  → 幻覚（Hallucination）や内容の浅さが課題となる

## ❖✅ OKパターン（台本生成をパイプライン化）

- **【分析 Phase】** README解析・技術スタック特定・ハイライト抽出など具体的に示して分析
- **【企画 Phase】** ターゲット設定・構成案（プロット）作成
- **【執筆 Phase】** セクションごとの台本生成
- **【構築 Phase】** Remotionコードへ渡すデータ構造（JSON）の生成
- **【検証 Phase】** データスキーマを検証



# AIを「挟み撃ち」にして品質を安定させる

「何をしてほしいか（Input）」と「何を返すべきか（Output）」を契約にして「ガチャ」を防ぐ

## ❖【分析フェイズ】リポジトリ分析もワークフロー化する

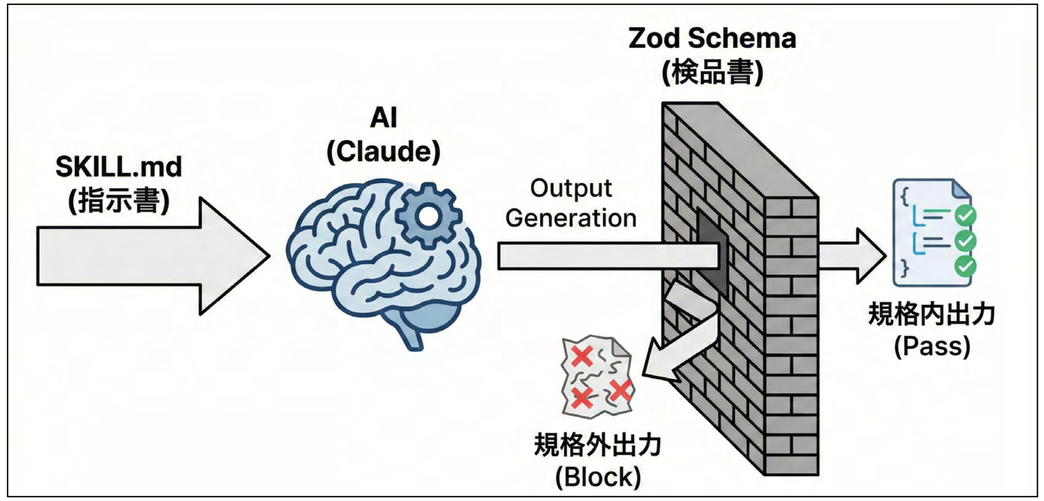
- Claude Code の Agent Skillsでリポジトリ分析処理を定型化している

### ✦入力の構造化

- SKILL.md で「**役割・手順・禁止事項**」を定義する
  - **役割**：「あなたはプロの動画クリエイターです」というペルソナと制約を与える
  - **手順**：リポジトリ分析処理の流れを定義する
    （必須情報の収集→技術スタック特定→ビジュアル要素の発見→プロジェクト構造→コミット履歴→ハイライトの抽出 など)
  - **禁止事項**：文字数制限、英文そのままでは使わない、など

### ✦出力の構造化（Schema 定義で検証）

- Zod Schema で「JSONの型・文字数・必須項目」を機械的に検証する
- 形式エラーがあれば、AIに自動で再生成させる
  （人間がやり直し指示しない）



04

# Part4. まとめ

# まとめ：AIに仕事を奪われるのではなく、AIの仕事を設定する

人間の役割は「Doing（作業）」から**「Directing（指揮）」と「Judging（審判）」**へ

## ❖AI駆動開発とは（再掲）

✦**単なるAIツールの導入ではなく、「開発の主体」と「人間の役割」の再定義である**

## ❖課題

- AI任せにすると「レビュ工数」だけが爆発し、品質や生産性は上がらない

## ❖解決策

- **ワークフロー定義・段階的依頼**：作業プロセスをルール化してAIが自律的に動ける仕組みを作る
- **構造化**：入出力を定義し、形式的なレビュを自動化する

## ❖結論

- AIに実施させるプロセス・ワークフローを定義できるスキルを磨く
- AIに正解を与えて仕事をさせる

# 宣伝

# 個人開発「Yukkuri Video Studio」で作成した動画を配信中！




# ご清聴ありがとうございました！

# END OF DOCUMENT