もうAIに振り回されない！OpenSpecで実現する予測可能なAI開発

AIコーディングアシスタントとの協業を次のレベルへ

アジェンダ

1. 現状の課題（Vibeコーディング）
2. 従来のアプローチ（CLAUDE.md/AGENTS.md）
3. 仕様駆動開発（SDD）とは
4. 従来のアプローチとOpenSpec比較
5. 他ツールとの比較
6. まとめ

自己紹介

吉永 章太朗

所属

サイオステクノロジー

PS/SL

技術領域

生成AI活用事業（RAG、Dify）

WEBアプリケーション開発

Vibeコーディングの実例

```
開発者: 「ユーザー認証機能を追加して」
AI:     [1000行のコード生成]

開発者: 「いや、JWTじゃなくてセッション認証で...」
AI:     [また1000行のコード生成]

開発者: 「そもそもReactじゃなくてNext.jsで...」
AI:     [さらに1000行のコード生成]
```

無限ループ地獄

😱

「Vibeコーディング」とは

Andrej Karpathy(OpenAI共同創業者)が2025年2月に初めて提唱

特徴

- LLMの性能向上で可能になった新しい開発手法
- コードの存在を忘れ、感覚的に開発する
- エラーメッセージはコメントなしでコピペ
- コードが開発者の理解を超えて成長
- 音声入力でAIに指示
- バグは修正できなければ回避策で対処するか、ランダムな変更で消えるまで試す

https://x.com/karpathy/status/1886192184808149383

根本的な4つの問題

1. 曖昧な要求仕様

「何を作るか」が明確でない → AIが推測で補完

2. コンテキストの欠如

プロジェクトの技術スタック、規約、アーキテクチャが不明

3. 非決定的な出力

同じプロンプトでも毎回違う結果

4. 知識の散逸

決定事項がチャット履歴に埋もれる

従来のアプローチ：CLAUDE.md

CLAUDE.mdとは？

プロジェクトのルートに配置する、**AIへの指示書**

- **役割**: Claude Code（Anthropic社のAIコーディングエージェント）に特化
- **目的**: プロジェクトのガイドライン、ベストプラクティス、制約を記述

CLAUDE.mdの例

```
# プロジェクト概要
ECサイトのバックエンド開発

## 技術スタック
- Node.js + TypeScript
- Express.js、PostgreSQL、Jest

## コーディング規約
- 関数は単一責任の原則に従う
- エラーハンドリングは必須
- テストファーストで実装

## 実装時の注意
1. TypeScriptの型定義を含める
2. APIはRESTfulに設計
3. 認証はJWT Bearer tokenを使用
```

CLAUDE.mdの詳細

包括的なプロジェクトガイド

CLAUDE.mdには以下の内容を含めることが推奨されます：

基本情報:

- プロジェクト概要と目的
- 技術スタック
- ディレクトリ構造

開発ガイド:

- コーディング規約（命名規則、ファイル構成）
- データモデル仕様
- APIエンドポイント一覧
- 認証フロー

運用情報:

- セキュリティ要件
- 開発ワークフロー
- テスト戦略
- トラブルシューティング

CLAUDE.mdの進化の歴史

第1段階（2025年6月頃）: チートシート

- プロジェクトの基本情報提供
- 共通コマンドのリスト化

第2段階（2025年7月〜9月）: AIの制約

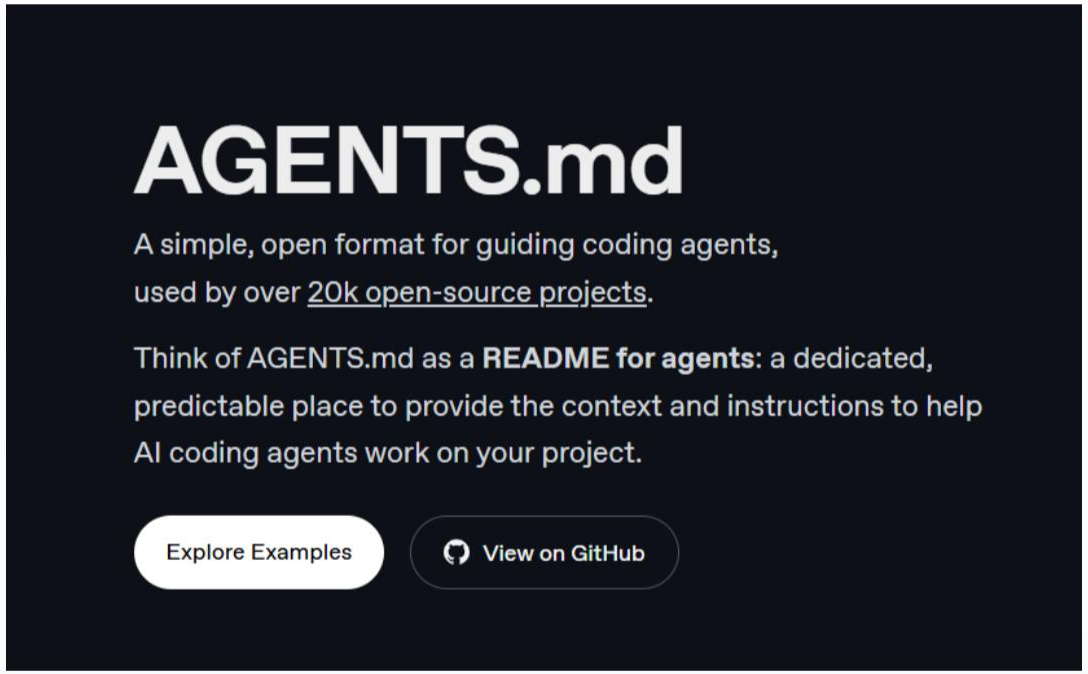
- 詳細なスタイルガイド
- ワークフローの自動化
- 失敗の記録と学習
- 階層的な設定管理（モノレポ対応）

AGENTS.mdの登場

複数のAIツール間で共有される共通指示

- **公開日**: 2025年8月19日（OpenAIによる）
- **目的**: オープンフォーマットとして標準化
- **利点**: Claude以外のAIツールでも利用可能

https://agents.md



AGENTS.mdの詳細

AIエージェント向け実装指示書

推奨言語: 英語（多様なAIツールとの互換性向上）

実装ガイドライン:

- コード規約（命名規則、ファイル構成、型定義）
- エラーハンドリングパターン
- セキュリティチェックリスト
- 認証実装パターン

ワークフロー指示:

- 新機能開発プロセス（要件確認→実装→検証→コミット）
- バグ修正プロセス
- コードレビュー基準

AGENTS.md：英語推奨の理由

なぜ英語で作成すべきか

1. **AIモデルの学習データ**: 英語が最も豊富
2. **技術用語の正確性**: 英語のほうが曖昧性が少ない
3. **国際的な互換性**: グローバルチームでの共有が容易

ベストプラクティス

```
# TODO Application - Agent Instructions

## Project Overview
**Name**: TODO List Application for Engineers
**Status**: In Development (v1.0)

## Development Workflow
### New Feature Implementation Process
...
```

CLAUDE.md/AGENTS.mdの限界（1）

1. 仕様の管理が困難

CLAUDE.md、 AGENTS.md

→ プロジェクト全体の説明が含まれている...

✗ すべての機能の仕様が混在している
✗ 過去の決定事項がどこかに埋もれている
✗ 現在の作業内容が見つけにくい
✗ 途中参加メンバーが全体像を把握することが困難

```
# TODO Application - Agent Instructions

This file serves as an instruction manual for AI agents working on this project.

## Project Overview

**Name**: TODO List Application for Engineers
**Status**: In Development (v1.0)
**Last Updated**: 2025-11-17

**Current Implementation Status**:
- Backend API (TODO CRUD operations)
- CosmosDB Integration
- Authentication (In Progress)
- Frontend UI (Not Started)

## Technology Stack

| Area | Technology |
|------|------|
| Backend Framework | NestJS 10 |
| Frontend Framework | Next.js 15 (App Router) |
| Language | TypeScript |
...
```

CLAUDE.md/AGENTS.mdの限界（2）

2. 変更の追跡が不可能

- どの機能がいつ追加されたか不明
- 仕様変更の履歴が残らない
- 複数人での作業時に混乱
- セッションが異なると情報が共有されない

3. スケーラビリティの欠如

- プロジェクトが大きくなると管理不能
- **既存機能の変更（1→N）が特に困難**
- 各フォルダにCLAUDE.md/AGENTS.mdが点在
- 全体像の把握が困難に

仕様駆動開発（SDD）の台頭

Kiro（AWS）がプレビューリリース (2025年7月15日)
「仕様駆動開発」を強く打ち出し

Spec Kit（GitHub）がリリース (2025年9月2日)
GitHub Copilotで実現するオープンソースツール

CLAUDE.md/AGENTS.mdの役割が再評価
仕様の明確化と共有の重要性が注目される

現在の仕様駆動開発ツール

主要なSDDツール（2025年11月時点）

- **Kiro** (AWS)
- **Spec Kit** (GitHub)
- **OpenSpec** (Fission AI) ← 本日紹介
- **BMad Method** (BMad Code)
- **cc-sdd** (国産OSSツール)

https://kiro.dev
https://github.com/github/spec-kit
https://github.com/Fission-AI/OpenSpec
https://github.com/bmad-code-org/BMAD-METHOD
https://github.com/gotalab/cc-sdd

仕様駆動開発（SDD）とは

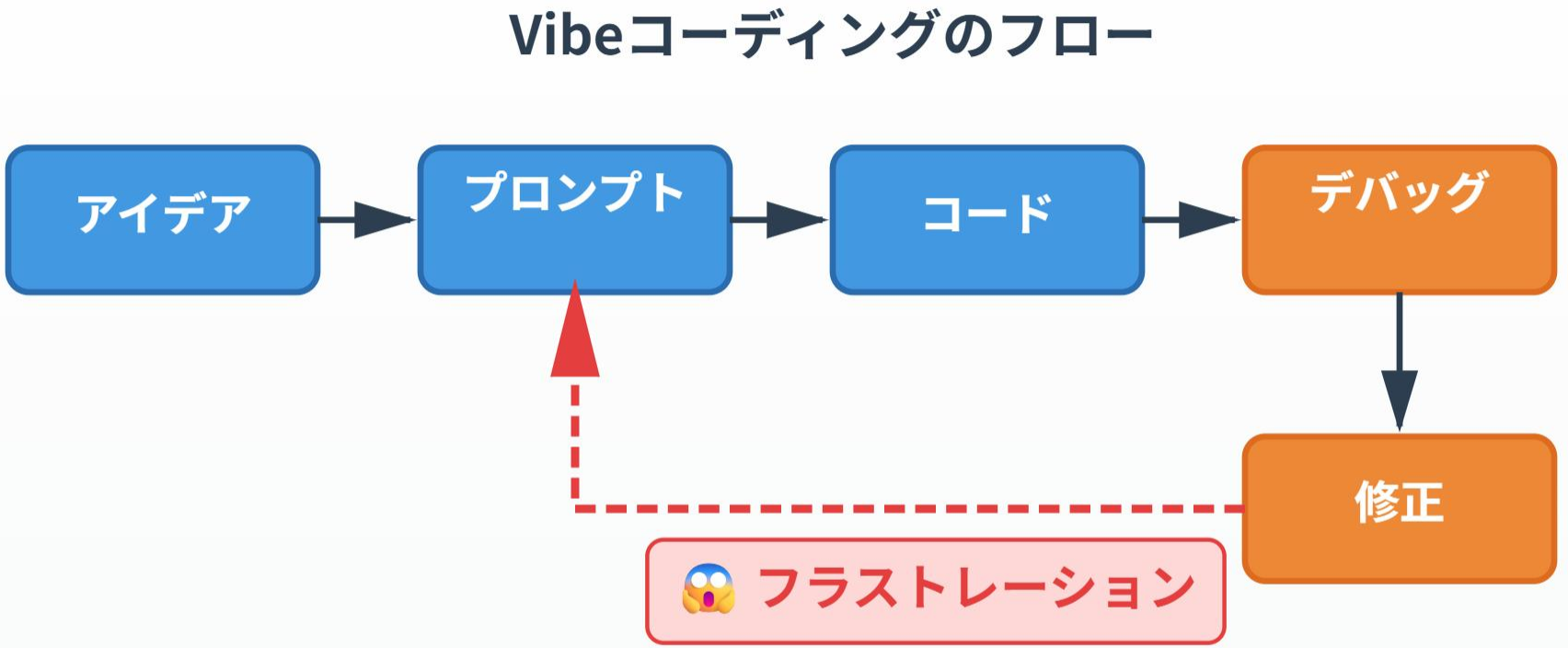
SDDの本質

> ***「コードを書く前に、何を作るかをAIと合意する」***

キーコンセプト

- **明確な仕様定義**：曖昧さを排除する
- **人間と AI の共通理解**：同じ認識を持つ
- **予測可能な実装**：実装前に結果が見える
- **追跡可能な変更**：いつ、何が変わったか分かる

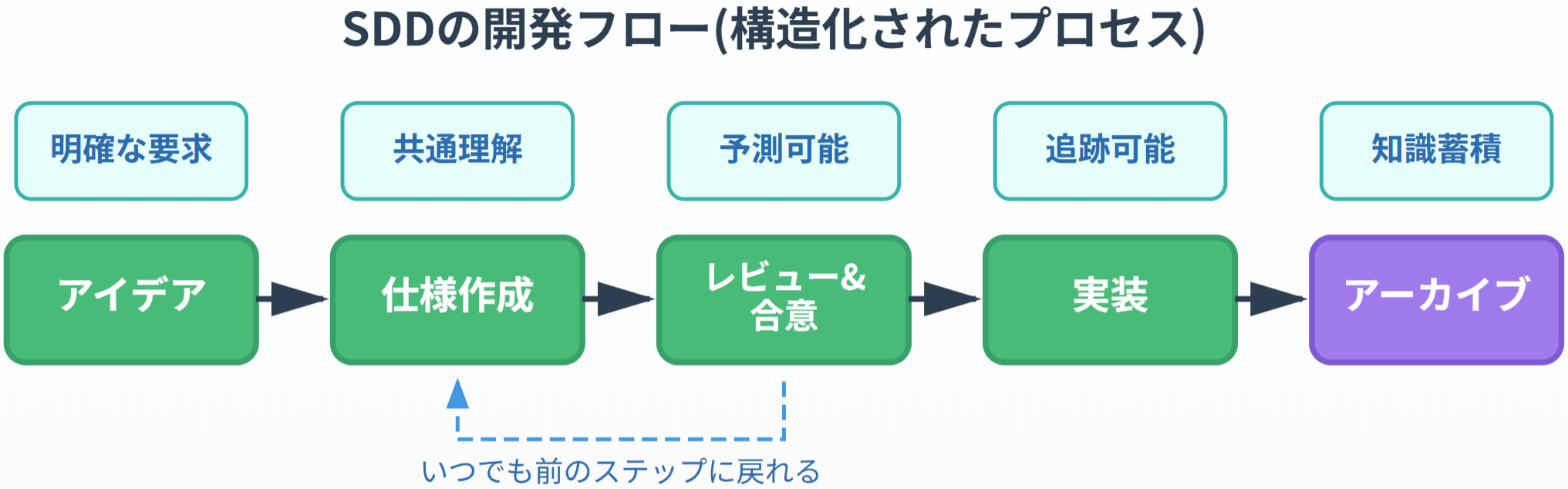
従来フローの問題



何が問題か？

- ❌ ゴールが不明確
- ❌ 無限ループに陥りやすい
- ❌ 手戻りが多い
- ❌ 予測不能
- ❌ 知識が蓄積されない

SDDの開発フロー



メリット:

- ✓各ステップが明確
- ✓いつでも前のステップに戻れる
- ✓決定事項が文書化される
- ✓チーム協業が容易
- ✓知識が組織に蓄積
- ✓予測可能な開発

SDDがもたらす4つの価値

1. 予測可能性

実装前に結果が予測できる

2. 監査可能性

すべての変更が追跡可能

3. 再利用性

仕様が知識として蓄積される

4. チーム協業

異なるAIツールでも同じ仕様を共有

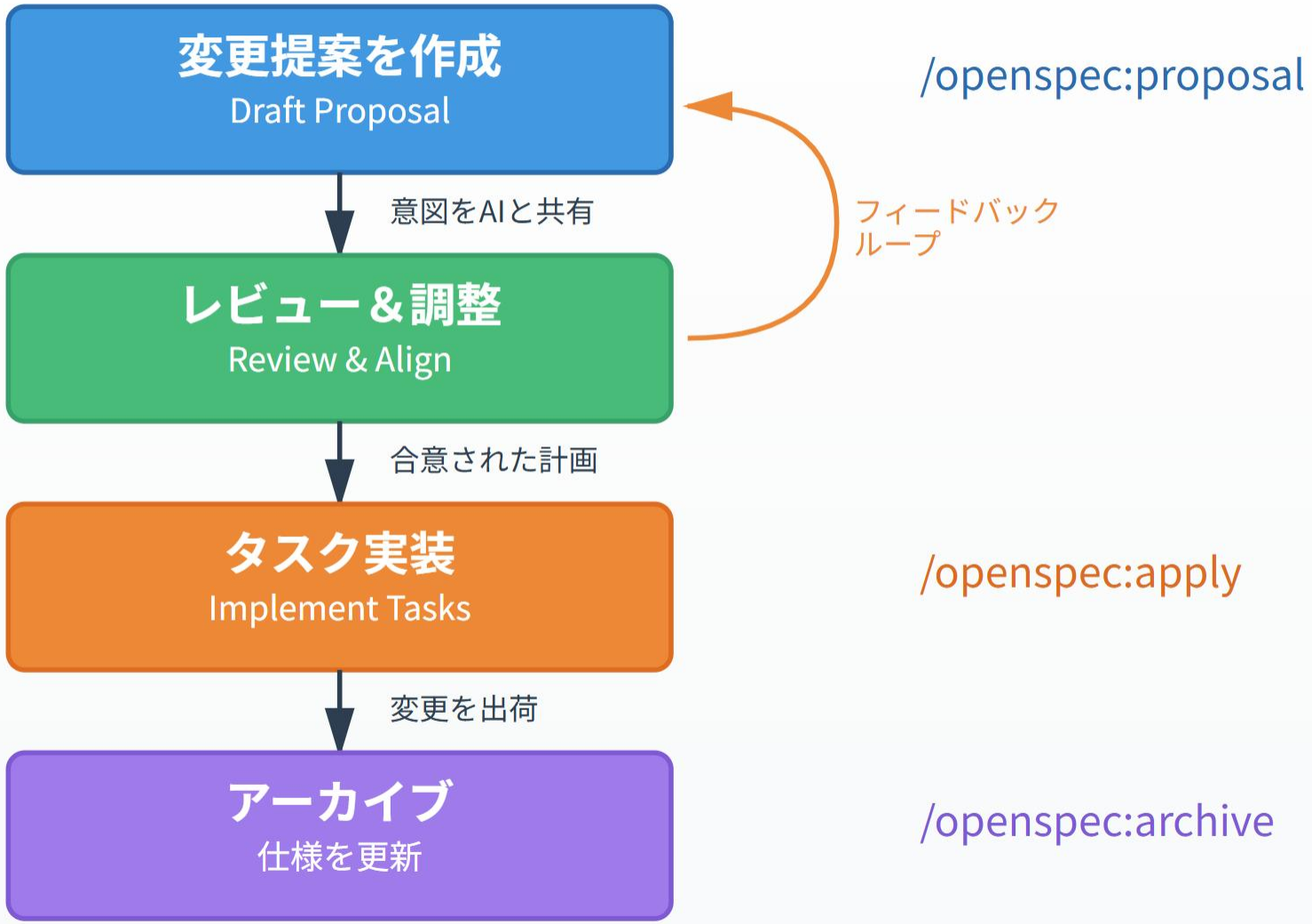
OpenSpecの登場

仕様駆動開発を実現する軽量なワークフロー管理ツール

- **開発**: Fission AI
- **特徴**: 既存プロジェクト向けに最適化
- **対応**: 複数のAIツール（Claude Code、Cursor、GitHub Copilotなど）

https://github.com/Fission-AI/OpenSpec

OpenSpecワークフロー



シナリオ：二要素認証（2FA）を追加

実際のケースで比較

目標: 既存の認証システムに2FAを追加する

- CLAUDE.mdのみの場合
- OpenSpecを使用した場合

CLAUDE.mdのみの場合

開発者の作業

1. CLAUDE.mdを手動で更新（どこに書くか迷う）
2. AIにプロンプト：「2FAを追加して」
3. 大量のコードが生成される
4. レビューが困難（何が変更されたか不明）
5. 既存機能への影響範囲が不明
6. 他の開発者には変更が伝わらない

結果

- × 仕様がCLAUDE.md内で埋もれる
- × 変更履歴が残らない
- × チーム間での共有が困難

```
# TODO List Application - Project Guide

このファイルはClaudeがプロジェクトを理解し、適切にサポートするための包括的なガイドです。

## プロジェクト概要

**目的**: 個人のエンジニア向けTODOリストアプリケーション

**主要機能**:
- Google OAuth認証によるセキュアなユーザー管理
- TODO の CRUD 操作（作成、取得、更新、削除）
- ステータス・優先度によるフィルタリング
- ユーザーごとのデータ完全分離

**ユーザー像**:
- 個人のソフトウェアエンジニア
- 技術レベル：中級〜上級
- 数百件のTODO管理を想定

## 技術スタック

### フロントエンド

Framework:      Next.js 15 (App Router)
UI Library:     React 19 + Shadcn/ui
Styling:        Tailwind CSS
State:          SWR for data fetching
```

OpenSpecを使用した場合

```
# Step 1: 提案作成
/openspec:proposal 二要素認証を追加

# Step 2: 生成されたファイル構造
openspec/changes/add-2fa/
├── proposal.md     # 変更の意図
├── tasks.md        # 実装タスク
├── design.md       # 技術仕様
└── specs/auth/
    └── spec.md     # 2FAの仕様

# Step 3: 実装
/openspec:apply add-2fa

# Step 4: アーカイブ
/openspec:archive add-2fa
```

Step 1: 変更提案の作成

```
開発者: 「プロフィール検索にロールとチームでのフィルター機能を
        追加するOpenSpec変更提案を作成して」

または: /openspec:proposal プロフィール検索フィルター追加

AI: 「OpenSpec変更提案を作成します...」
    *openspec/changes/add-profile-filters/を生成*
```

生成されるもの

- proposal.md（変更の意図と背景）
- tasks.md（実装タスクリスト）
- specs/（仕様の差分）

Step 2: 仕様のレビューと調整

```
開発者: 「ロールとチームフィルターの受け入れ条件を追加して」

AI: 「仕様差分を更新します...」
    *specs/profile/spec.mdとtasks.mdを編集*
```

ポイント

- 人間が仕様をレビュー
- AIと対話しながら詳細化
- 合意が得られるまで繰り返し

Step 3: 実装

```
開発者: 「仕様が良さそうです。実装しましょう」

または: /openspec:apply add-profile-filters

AI: 「タスクを順番に実装します...」
    Task 1.1 ✓ データベーススキーマ更新
    Task 1.2 ✓ APIエンドポイント追加
    Task 2.1 ✓ フロントエンドコンポーネント作成
```

利点

- タスクが段階的に完了
- 進捗が可視化される
- 問題があればすぐに気づける

Step 4: アーカイブと知識の蓄積

```
 開発者: 「変更をアーカイブして」

または: /openspec:archive add-profile-filters

 AI: 「変更をアーカイブします...」
    *openspec archive add-profile-filters --yes*
    ✓ 仕様が更新されました
```

結果

- specs/フォルダに仕様が統合される
- changes/フォルダから変更が削除される
- 知識が組織に蓄積される

核となる概念：2つのフォルダ

```
AGENTS.md
openspec/
├── specs/              # 現在の仕様
│   └── auth/
│       └── spec.md # 認証機能の仕様
│
└── changes/            # 提案中の変更
    └── add-profile-filters/
    |   ├── proposal.md   # 変更内容
    |   ├── tasks.md      # タスクリスト
    |   ├── design.md     # 技術仕様
    |   └── specs/
    |       └── auth/
    |           └── spec.md # 仕様の差分
    └── add-frontend-ui/
...
```

変更差分の概念

GitのPull Requestと同様の考え方

```
# プロフィール仕様の変更差分

## ADDED Requirements
### 要件: ロールフィルター
システムはユーザーロールによるフィルタリングを提供しなければならない

## MODIFIED Requirements
### 要件: 検索レスポンス（更新版）
検索結果にはフィルター適用状態を含めなければならない

## REMOVED Requirements
### 要件: 全件表示
（廃止：パフォーマンスの問題により）
```

変更差分のメリット

レビュー時に一目瞭然

- **ADDED** - 新機能
- **MODIFIED** - 変更
- **REMOVED** - 廃止

コードレビューとの類似性

- 差分が明確
- 変更の影響範囲が分かる
- 承認プロセスが容易

```
# TODO API Specification

## ADDED Requirements

### Requirement: TODO Creation

システムはTODOを作成する機能を提供しなければなりません(MUST)。

#### Scenario: 必須フィールドのみでTODO作成成功

- **WHEN** ユーザーが title と status を含む有効なTODOデータをPOSTする
- **THEN** システムは新しいTODOを作成し、201 Createdステータスとともに作成されたTODOを返す
...
## MODIFIED Requirements

#### Scenario: 必須フィールドが欠けているTODO作成失敗

- **WHEN** ユーザーが title なしでTODOデータをPOSTする
- **THEN** システムは400 Bad Requestステータスとバリデーションエラーメッセージを返す

# REMOVED Requirements
...
### Requirement: TODO Deletion

システムはTODOを削除する機能を提供しなければなりません(MUST)。

#### Scenario: TODO削除成功

- **WHEN** ユーザーが有効なTODO IDを指定してDELETEリクエストを送信する
- **THEN** システムは指定されたTODOを削除し、204 No Contentステータスを返す
...
```

既存のAGENTS.md/CLAUDE.mdは不要？

- 併用が推奨されている
- AGENTS.md/Claude.mdは全体的な指示、OpenSpecは個別機能の仕様管理

```
<!-- OPENSPEC:START -->
# OpenSpec Instructions

These instructions are for AI assistants working in this project.

Always open `@/openspec/AGENTS.md` when the request:
- Mentions planning or proposals (words like proposal, spec, change, plan)
- Introduces new capabilities, breaking changes, architecture shifts, or big performance/security work
- Sounds ambiguous and you need the authoritative spec before coding

Use `@/openspec/AGENTS.md` to learn:
- How to create and apply change proposals
- Spec format and conventions
- Project structure and guidelines

Keep this managed block so 'openspec update' can refresh the instructions.

<!-- OPENSPEC:END -->
```

```
<!-- OPENSPEC:START -->
# OpenSpec 操作手順

この手順は、本プロジェクトで活動するAIアシスタント向けです。

以下のリクエスト時には常に `@/openspec/AGENTS.md` を開いてください：
- 計画や提案に関する言及がある場合（proposal、spec、change、plan などの単語を含む）
- 新機能の導入、互換性のない変更、アーキテクチャ変更、大規模なパフォーマンス/セキュリティ作業に関するもの
- 曖昧な内容で、コーディング前に正式な仕様書が必要な場合

`@/openspec/AGENTS.md` で以下の内容を学習してください：
- 変更提案の作成と適用方法
- 仕様書のフォーマットと規約
- プロジェクト構造とガイドライン

この管理ブロックを維持し、「openspec update」で手順を更新できるようにしてください。

<!-- OPENSPEC:END -->
```

Kiro：プロジェクト管理の実例

プロジェクト構造

```
.kiro/
├── steering/              # プロジェクト方向性（変更頻度：低）
│   ├── product.md         # プロダクトのビジョン、目標、ターゲットユーザー、主要機能
│   ├── tech.md            # 技術スタック、アーキテクチャ方針
│   └── structure.md       # ディレクトリ構造、命名規則
└── specs/                 # 機能仕様（変更頻度：高）
    └── add-2fa/
        ├── requirements.md  # 機能要件
        ├── design.md        # 技術スタック、アーキテクチャ方針
        └── tasks.md         # 実装タスク
```

Kiroの2層構造

- **steering/** - プロジェクト全体の方向性（変更頻度：低）
- **specs/** - 個別機能の仕様（変更頻度：高）

Kiro：steering/の役割

プロジェクト全体の方向性を3つのファイルで管理

product.md - プロダクトの目的と主要機能:

```
## Purpose
エンジニア向けTODOリスト

## Key Features
- シンプルなTODO管理
- Git/GitHub連携
- Google OAuth認証

## Target Users
エンジニア
```

Kiro：steering/tech.md

技術スタックの定義

tech.md - フレームワークとライブラリの選定:

```
## Languages & Frameworks
- Next.js 15
- Tailwind CSS
- NestJS

## Key Libraries
- Shadcn/ui

## Common Commands
### Development
`npm run dev`

### Testing
`npm test`
```

Kiro：steering/structure.md

構造ガイドライン

structure.md - プロジェクト構造とアーキテクチャ:

- ディレクトリ構成
- 命名規則
- アーキテクチャパターン
- モジュール間の依存関係

steering/の特徴

- 変更頻度が低い（プロジェクトの基盤）
- 全機能に共通するルール
- 新メンバーのオンボーディング資料

Kiro：specs/の仕様書構造

requirements.md - 機能要件の定義

```
### 要件1: ユーザー認証

**ユーザーストーリー:**
エンジニアとして、セキュアにアカウントにアクセスしたい

#### 受入基準
1. WHEN ユーザーがログインボタンをクリックしたとき、
   TODOシステムはGoogle OAuth認証フローを開始すること
2. WHEN Google認証が成功したとき、
   TODOシステムはユーザーセッションを作成すること
```

Kiro：design.mdとtasks.md

design.md - アーキテクチャ設計

```
# 設計書

## 概要
本システムは、Next.js 15をフロントエンド、NestJSをバックエンドとして...

### 技術スタック

**フロントエンド:**
- Next.js 15 (App Router)

## アーキテクチャ

### プロジェクト全体構造
application/
├── backend/                    # NestJS バックエンド

## APIエンドポイント設計

### 認証エンドポイント
GET  /api/auth/google          # Google OAuth開始
```

tasks.md - 実装タスク

```
## Phase 1: 基盤構築
- [ ] プロジェクトセットアップ
- [ ] 認証システム実装

## Phase 2: 機能実装
- [ ] TODO CRUD操作
- [ ] GitHub連携
```

Kiroの2層構造の利点

steering/とspecs/の分離

steering/（低頻度変更）：

- プロジェクトの憲法
- すべての機能に影響
- 変更は慎重に

specs/（高頻度変更）：

- 個別機能の詳細
- 独立して変更可能
- アジャイルな開発

メリット

- 全体方針と個別機能を分離
- 大規模プロジェクトに適している
- チーム間の役割分担が明確
  - PM: steering/
  - 開発者: specs/

Spec Kit：プロジェクト構造

specs/ディレクトリ - 機能仕様の管理

```
specs/001-todo-app-git-oauth/
├── spec.md                # 機能仕様書
├── checklists/
│   └── requirements.md  # 品質チェックリスト
├── plan.md                # 実装計画（技術選定）
├── research.md            # Phase 0: 技術調査
├── data-model.md          # Phase 1: データモデル
├── contract
│   └── openapi.yaml       # Phase 1: API制約
├── tasks.md               # Phase 2: 実装タスク
└── quickstart.md          # 開発者ガイド
```

Spec Kit：4フェーズワークフロー

段階的な開発プロセス

- **Phase 0**: 仕様作成（spec.md + research.md） → 何を作るかを定義
- **Phase 1**: 設計（plan.md + data-model.md + contracts/）→ どう作るかを設計
- **Phase 2**: 実装（tasks.md） → 具体的な実装ステップを実行
- **Phase 3**: レビュー（quickstart.md） → 実装内容の動作確認

各フェーズで明確な成果物を作成

新規プロジェクトを0から立ち上げる際の完全なガイド

Spec Kit：Phase 0 - 仕様作成

spec.md - 機能仕様書

```
## Feature: ユーザー認証システム

### User Story
エンジニアとして、安全にアプリケーションにアクセスしたい

### Acceptance Criteria
- Google OAuthでログイン可能
- セッションは24時間有効
- ログアウト機能を提供
```

research.md - 技術調査

ライブラリの比較、セキュリティ要件の調査など

Spec Kit：Phase 1 - 設計

plan.md - 実装計画

- 技術選定の理由
- アーキテクチャ方針
- 開発スケジュール
- リスク管理

data-model.md - データモデル

```
User:
  - id: UUID
  - email: string
  - name: string
  - createdAt: timestamp
```

contract/openapi.yaml

API仕様をOpenAPI形式で定義

- 型安全性を確保
- 自動でAPI仕様ドキュメントを生成可能

Spec Kit：Phase 2-3 - 実装とレビュー

Phase 2: tasks.md - 実装タスク

```
## Backend Setup
- [ ] プロジェクト初期化
- [ ] データベース設定
- [ ] 認証エンドポイント実装

## Frontend Setup
- [ ] ログインUI作成
- [ ] セッション管理実装
```

Phase 3: quickstart.md - 開発者ガイド

ガイドの内容:

- 環境構築手順
- 開発サーバー起動方法
- テスト実行方法
- デプロイ手順

新メンバーのオンボーディングに最適

Spec Kit：品質チェックリスト

checklists/requirements.md

品質保証のための詳細なチェック項目:

- **機能要件**: すべての機能が実装されているか
- **非機能要件**: パフォーマンス、セキュリティ、可用性
- **テストカバレッジ**: ユニット、統合、E2Eテスト
- **ドキュメント**: README、API仕様、ユーザーガイド
- **コード品質**: リント、フォーマット、レビュー

特徴

各フェーズでチェックリストを確認し、品質を担保

Spec Kit：新規プロジェクト向けの強み

0→1に最適化

- プロジェクト立ち上げの完全なガイド
- 重厚なドキュメント構成
- 品質チェックリスト完備
- quickstart.mdで即座に開発開始

メリット

- すべてのフェーズが明確
- 新メンバーのオンボーディングが容易
- OpenAPI連携で型安全性を確保
- GitHub Copilotとの統合

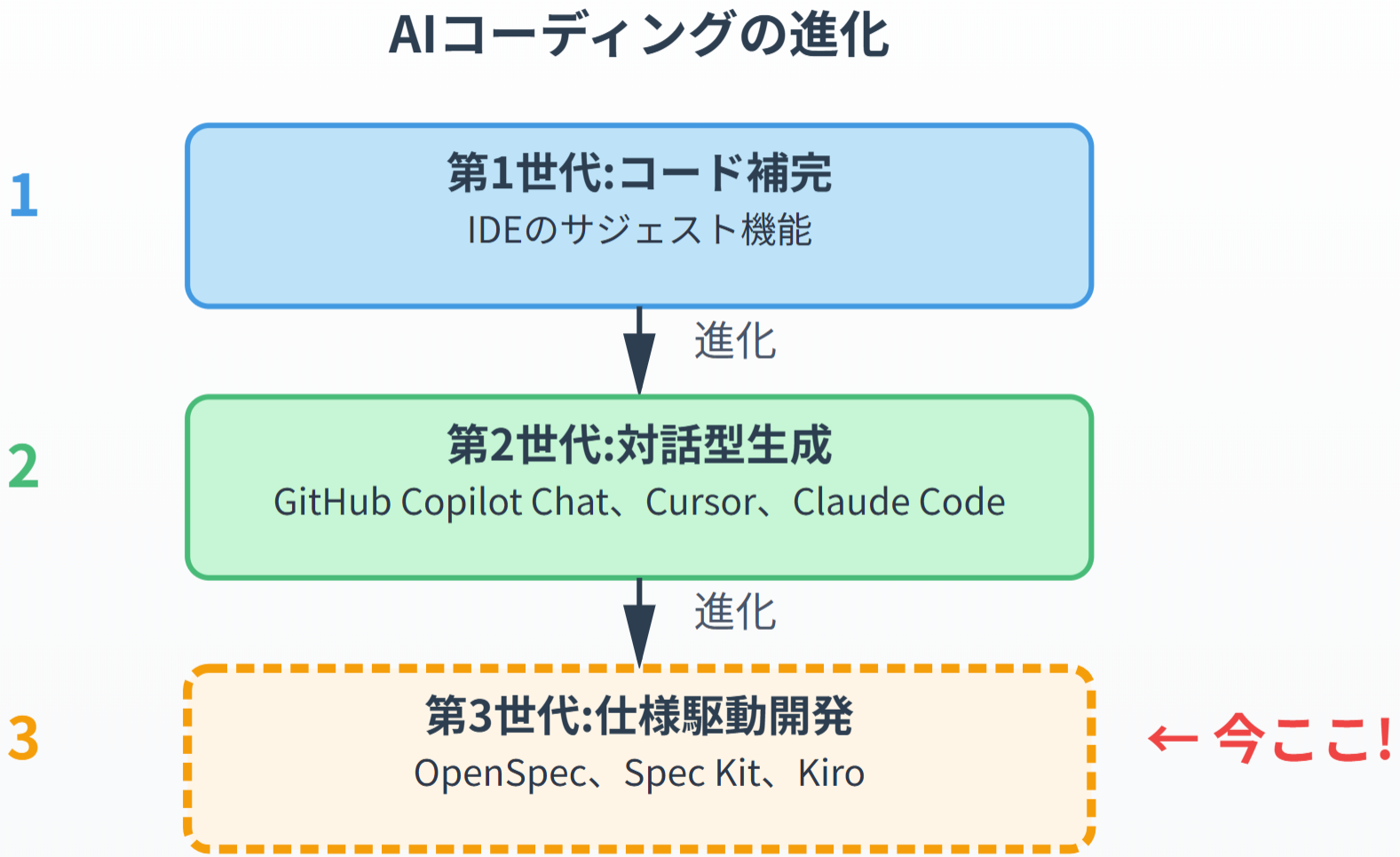
主要なSDDツールの比較

特徴	OpenSpec	Spec Kit	Kiro
対象	既存PJ (1→N)	新規PJ (0→1)	全プロジェクト
構造	2フォルダ (specs + changes)	4フェーズファイル	3フェーズファイル
主要ファイル	proposal.md tasks.md spec 差分	spec.md + research.md plan.md + data-model.md tasks.md quickstart.md	requirements.md design.md tasks.md
変更管理	差分 + アーカイブ	Git履歴	Git履歴
価格	無料（AIツール利用料のみ）	無料（AIツール利用料のみ）	有料（毎月 無料枠あり）

SDDツールのまとめ

- **OpenSpec**：
  既存プロジェクトに導入したいなら、これ。軽量で学習コストが低く複数の変更を並行して進めやすい。
- **Spec Kit**：
  新規プロジェクトを立ち上げるなら、これ。0 から丁寧にガイドしてくれる。ドキュメントが重厚。
- **Kiro**：
  大規模で長期的なプロジェクトでしっかりした構造が必要なら、これ。steering と specs の2層構造が強力。

AIコーディングの進化



仕様駆動開発がもたらす変化

開発者の役割の変化

- **Before**: AIのサポートでコードを書く
- **After**: 仕様を設計し、AIを操縦する

> - コードを読む能力、理解する能力は依然として重要
> - コードを書く作業は AI に任せていく
> - 「何を作るべきか」を明確に定義する能力が求められる

チーム開発の改善

- 知識の蓄積と共有

> - 仕様が残るので「なぜこうなっているか」が後から分かる
> - 新しいメンバーもキャッチアップしやすい

品質の向上

- 予測可能な出力
- レビュー可能な変更
- 追跡可能な履歴

まとめ

1. 仕様は新しいソースコード

AIとの協業において、明確な仕様が最も重要

2. AIとの協業には構造が必要

Vibeコーディングから脱却し、予測可能な開発へ

3. AI に振り回される開発から、AI を使いこなす開発へ

仕様駆動開発（SDD）でより良いソフトウェアを効率的に構築