AWS Summit Japan 2026

レガシーシステムのモダナイゼーション
ー Kiroを用いたAI駆動開発の実践ノウハウ

2026/6/25,26

豊岡　洋人

株式会社JSOL
データ＆テクノロジーコンサルティング事業本部
NTT DATA Group

Agenda

- 自己紹介
- はじめに
- Kiroとは
- 基盤連携でのユースケース
- 自社の検証内容・結果
- まとめ

自己紹介

氏名	豊岡　洋人 （とよおか　ひろと）
所属・役割	株式会社JSOL　データ＆テクノロジーコンサルティング事業本部　クラウドテクノロジー部
取り組み	AWS技術を利用したデータ活用・AI基盤構築など、 お客様のデータ・AI活用の全般的な支援を担当しています。 ＜受賞歴＞ 2025 Japan AWS Top Engineers 2025 Japan AWS All Certifications Engineers

JSOL関連サービス紹介　J-DAP（データ活用アクセラレーションプログラム）｜株式会社JSOL

はじめに

この話、あなたの現場でのAI活用で心当たりはありますか？

仕様の暗黙知化

生成したコードが唯一の仕様書。
読み解くだけで2週間かかる。

依存関係のブラックボックス化

基幹連携を触ると何が壊れるか
わからない。
だから誰も動けない。

コンテキスト不足による品質低下

生成AIに社内規約も設計意図
も渡せない。
出力が使えない。

この3つの話は、突き詰めると同じ原因に行き着く

はじめに

仕様が人の頭の中にしかなく AIに渡せるコンテキストがない

>>

生成物が社内標準・設計意図から外れる

>>

レビューコスト増・手戻り発生

"毎回指示するから仕様がどこにも残らない。
次の人が同じ問題にぶつかる

"生成されたコードが社内標準に合っていない。
ブラックボックスが増えただけ

"レビューが増えて、むしろ工数が増えた。
生成内容を上司に説明できない

AIは使えないという結論

ここまでの因果関係は「Vibe Coding」と呼ばれる開発スタイルに共通する。
鍵は、コード生成の前に仕様を構造化できるかどうか

Kiroとは

- 仕様を開発の中心に据えるために設計されたIDE
  - アマゾン ウェブ サービス（AWS）が提供するAI駆動開発支援ツール
  - 要件定義→設計→実装計画と段階的に仕様をすり合わせながら開発するため、**人とAIの認識のずれが発生しづらい**
  - **仕様と開発を同一環境で扱う**ことで、AIが常に仕様を参照でき、品質を担保しやすい

項目	従来のVibe Codingの課題	Kiroによる解決手段
仕様の明確化	PJの背景や指示の意図をAIが把握しておらず、想定した通りに実装されない	Requirements(要件)/Design(設計)/Tasks(実装計画)を定義したSpecファイルを実装前に作成し、要件の認識を合わせたうえで実装を開始
品質担保	とりあえず動くものはできるが、テスト結果や設計書等のドキュメントが残らず品質が担保されない	テスト実施やドキュメント作成も要件に含めることで、コード作成と合わせて実行
拡張性と統合	外部ツールとの連携が困難	MCP連携で外部知識・ツールを簡単に統合 また、VSCodeとの互換性があり拡張機能もそのまま利用可能
開発標準	使用する技術スタック、フォルダ構造、命名規則等のPJ特有の開発標準に準拠するのが困難	steeringにルールやプロダクトの概要を記載することで開発標準に準拠しながら実装

Kiroの開発フロー

Steering　PJ概要 ／ 技術スタック ／ コード規約 ／ セキュリティポリシー ／ ドキュメントテンプレート

↓ 常に参照

Specs

Requirements（要件定義）	Design（技術設計）	Tasks（実装計画）
プロンプトからユーザーストーリーと受入基準をEARS記法で構造化	システム構成・API設計 データフロー・処理ロジック を具体的に整理	依存関係に基づいて 順序付けされたタスク 一覧を自動生成

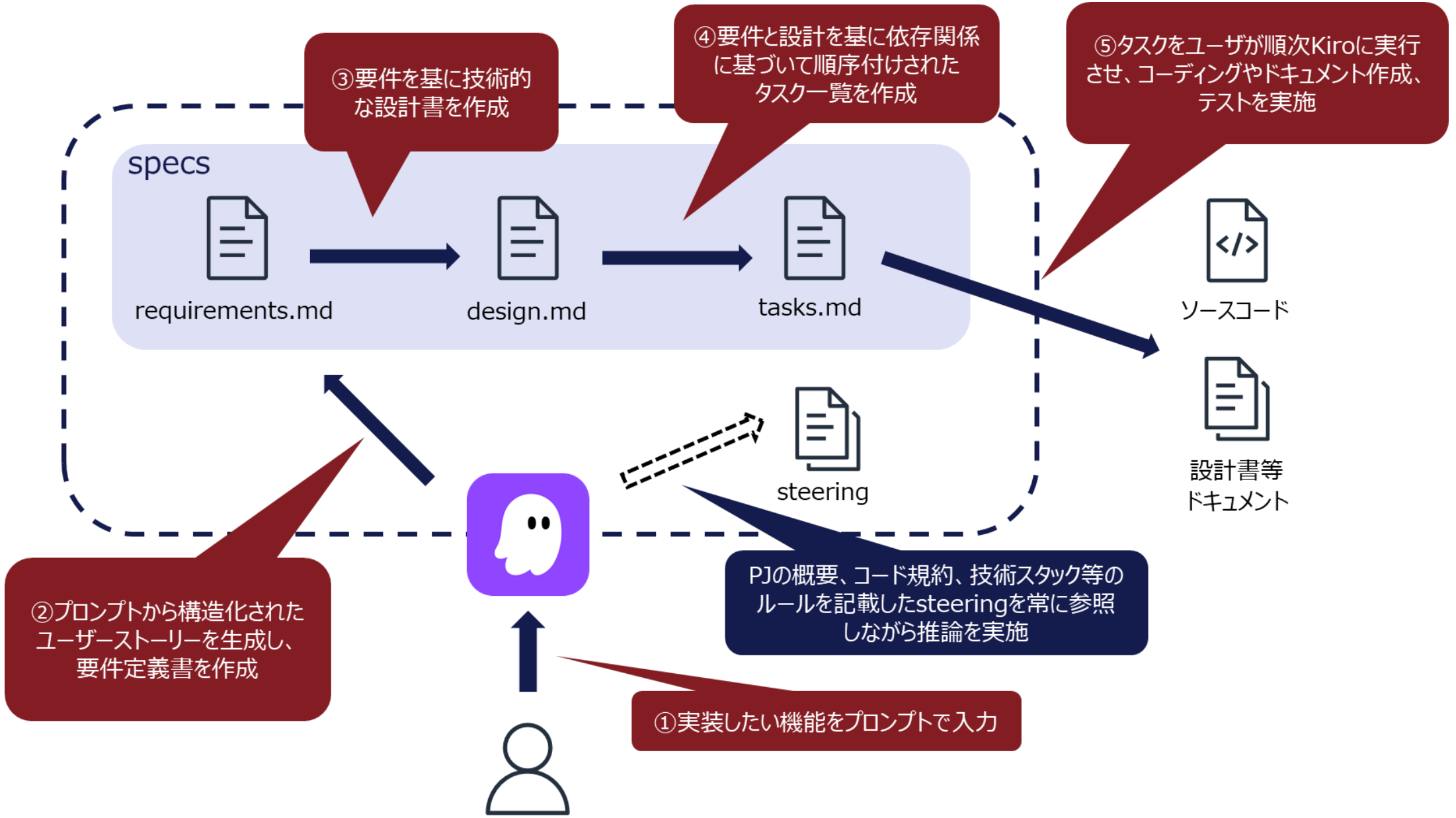
>>

↓

- ソースコード
- 画面設計書（HTML）
- API設計書（YAML）
- テスト仕様書（MD）
- 自動テストコード

各ステップで「何が残るか」が明示される ― これがVibe Codingとの本質的な違い

Kiroの開発フロー


③要件を基に技術的な設計書を作成
④要件と設計を基に依存関係に基づいて順序付けされたタスク一覧を作成
⑤タスクをユーザが順次Kiroに実行させ、コーディングやドキュメント作成、テストを実施
specs
requirements.md
design.md
tasks.md
ソースコード
設計書等ドキュメント
steering
②プロンプトから構造化されたユーザーストーリーを生成し、要件定義書を作成
PJの概要、コード規約、技術スタック等のルールを記載したsteeringを常に参照しながら推論を実施
①実装したい機能をプロンプトで入力

基幹連携コードをKiroに読ませたら

Before：レガシーコードの現実

- 仕様書が存在しない、またはメンテされていない
- 属人化した実装で、担当者以外が理解できない
- テストコードがなく、改修の影響範囲が見えない
- 基幹システム連携のロジックが複雑に絡み合い
  コードリーディングだけで数週間かかる

>>

Kiroで
モダナイズ

After：Kiroによる段階的モダナイゼーション

Step 1	既存コードをKiroに読み込ませ、 アーキテクチャ・処理フローを自然言語で出力
Step 2	Specsで仕様を再構築し、 暗黙知をドキュメントとして形式化
Step 3	テストコードを自動生成し、 改修前の振る舞いを保証する安全網を構築
Step 4	全面刷新ではなく、優先度の高い領域からAPIで切り出す。 基幹連携は維持したまま段階的に分離

まず既存の構造を理解し、テストで保護してから変更する。
この順序であれば基幹連携を壊さずにモダナイゼーションを進められる

自社でのKiro実証内容

項目	内容
対象システム	既存の業務Webアプリケーション（Python/Django ＋ JavaScript）
検証テーマ	機能追加フェーズ（6機能）をKiroで実施し、通常開発と工数を比較
検証スコープ	機能設計 → 実装 → 単体テスト → 結合テスト
比較方法	同一要件に対し、通常開発チームとKiroチームが並行して開発。実測工数を比較
除外項目	管理工数（打合せ・レビュー）、ツール習熟コスト

検証対象機能

画面追加	ファイル取込 ＋DB登録	データ 可視化	データ更新 ＋再描画	演算処理 ＋可視化	帳票出力
難度：低	難度：中	難度：中	難度：中	難度：高	難度：低

検証のポイント

- 新規開発ではなく、既存コードベースへの追加開発。実際のPJで発生する制約（既存設計への準拠、DB整合性）を含む
- フロントエンドのみの軽微な改修から、業務固有アルゴリズムを含む高難度機能まで幅広くカバー
- Kiro検証チームは通常開発チームとは別メンバーで構成（既存PJへの習熟度の差も含めた比較）

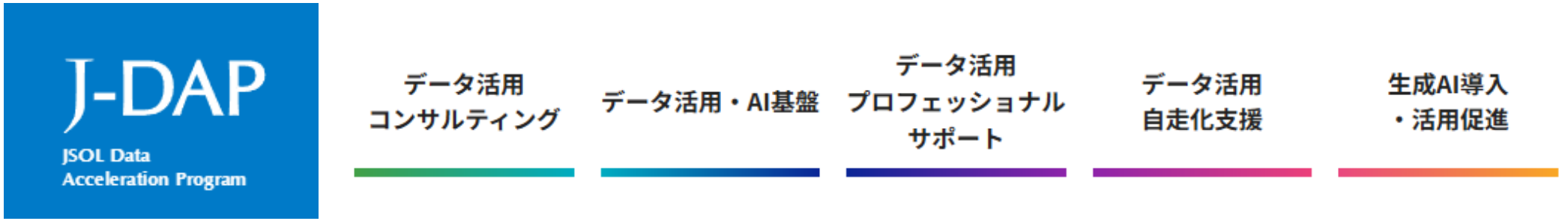
Kiro実証結果

通常開発比　58%　工数削減を実証

工程	通常開発 工数(h)	Kiro活用 工数(h)	増減	備考
設計	39.5	52.5	**33%増**	Steering作成は初回のみ発生する先行投資。 加えて、通常開発チームが持つ既存ドメイン知識の差も含まれている
実装	209	25.5	**88%削減**	コーディング・自動テスト含む
テスト	142	86.5	**39%削減**	仕様書作成＋手動テスト含む
合計	**390.5**	**164.5**	**58%削減**	管理工数・習熟コストは除外

ご清聴ありがとうございました。

今回ご紹介した「データ活用を加速する組織・環境構築」や、
その他のデータ活用に関するお問合せは弊社ソリューションページからお問合せください。



J-DAP（データ活用アクセラレーションプログラム） | 株式会社JSOL

今はない、答えを創る。

JSOL

NTT DATA Group