

AI SHOWCASE LT

コードの99%をAIが書く ネットワーク運用部署の、次の課題

「SDLC（開発ライフサイクル）のガバナンス」をどうAIに委ねるか

ソニーネットワークコミュニケーションズ
Network Automation Specialist
渡部友也

自己紹介

— “開発が得意ではない世界”に転生したけど、AIと一緒にもがいている人

渡部友也（わたなべ ともや）

ソニーネットワークコミュニケーションズ 技術部門ネットワーク部技術2課 兼 運用課
兼 OpenFiber Japan株式会社 技術本部ネットワーク開発2部ネットワーク構築2課

略歴

- 1985年 足立区生まれ
- ～2010年 都立航空高専 → 東京農工大 → 同大学院卒
- 2010～2024年 某Sierでプリセールス&NW自動化開発
- 2025年～ SNCにてNW自動化スペシャリストとして転生
(PON / 業務整理 / NRE / AI推進 / など)

趣味

- 🐎 競馬 (1998年～)
- 🀄 麻雀
- Ⓜ 広島カープファン歴30年
- 🍷 飲み歩き 新橋・上野・北千住



勉強会も主催しています

Network Developer Night — ネットワーク開発者の勉強会 (直近: #4 / 2026-07-23)

今日お話しするAI活用方法

「AIで 自動化コードを書く」から 「AIで 自動化プロジェクトを推進する」へ

人間の仕事は、下流から上流へ AIに委ねて いく 🤝

開発力ほぼゼロ



とりあえず書かせる



ガバナンスの強化



AIに推進してもらおう ▶ いまここ

※ なお、このスライド的なものも Quarto + reveal.js で Claude Opus 4.8 が作っています

GitHub Metricsで見る SNCネットワーク部の2025年～

15,966

総コミット

2,493

マージされたPR

179

累積リリース

1 → 25

AIアクティブユーザー / 月

🔄 変わったこと

増えたのはコミットだけじゃない——人もPRも、リリースも

→ [snc-nw](#) orgで蓄積したデータ。これらの変遷を3つの章に分けてお伝えします 🙌

スタート地点：開発力、ほぼゼロ

— 2025年1月 — 自動化活動を立ち上げた直後の現実

ネットワーク部のチームで、コードを書ける人は、ほぼいなかった。

3 / 45

コードを書けた社員

0

即戦力の委託メンバー

▶ 出発点

教材も時間もなく、メンバーへの
プログラミング教育は現実的ではない

→ だったら——AIに書かせるしかない

最初の打ち手：AIに“ただ書かせる”

— 2025年2～3月 — まだ GitHub Copilot Agent Mode が Preview だった頃

人が習熟して書くのは、やめた。**AIが書く** に 全振り した。

当時のAIコーディング事情 — 環境はまだ未熟だった

● Model は GPT-4.5 / Claude 3.7 Sonnet

● Claude Code はまだ無い

● “Vibe Coding” も未登場

手放した Python/TypeScriptを学習する

注力した Git/DevContainer等の周辺技術知識

👤 戦略

ネットワークエンジニアが、**プログラマになる必要はない**

→ 必要なのは「書く力」ではなく、AIに 書かせる力 と 業務知識

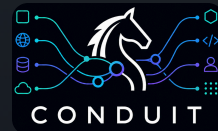
成功体験：システムが爆速で作られた

— 2025年～ — NRE活動として、ネットワーク運用ができるところからプログラム化



装置増設設計と作業

Netmiko Jinja



装置共通
Library/API Server

FastAPI Netmiko/Scrapli ttp/textFSM



2-way工事の自動化

Twilio React



IP/装置/構成管理

React Django



装置への一括操作
Orchestrator

Nornir

 最初の到達点

コードの**99%**は、**AIが書くことになった**
限られた開発者・適用範囲の中で、システムが複数できあがった

挫折：開発者と適用範囲がスケールしない

— 2025年2～8月（7ヶ月） — コミットは増えた。でもPRがマージされない

コードはできている

PRがマージされない

2,675

コミット
(月平均 ≈ 380)

34

立ち上げたりポジトリ
(～2025/8・次々と新規)

のに

173

マージPR数
(月平均 ≈ 25)

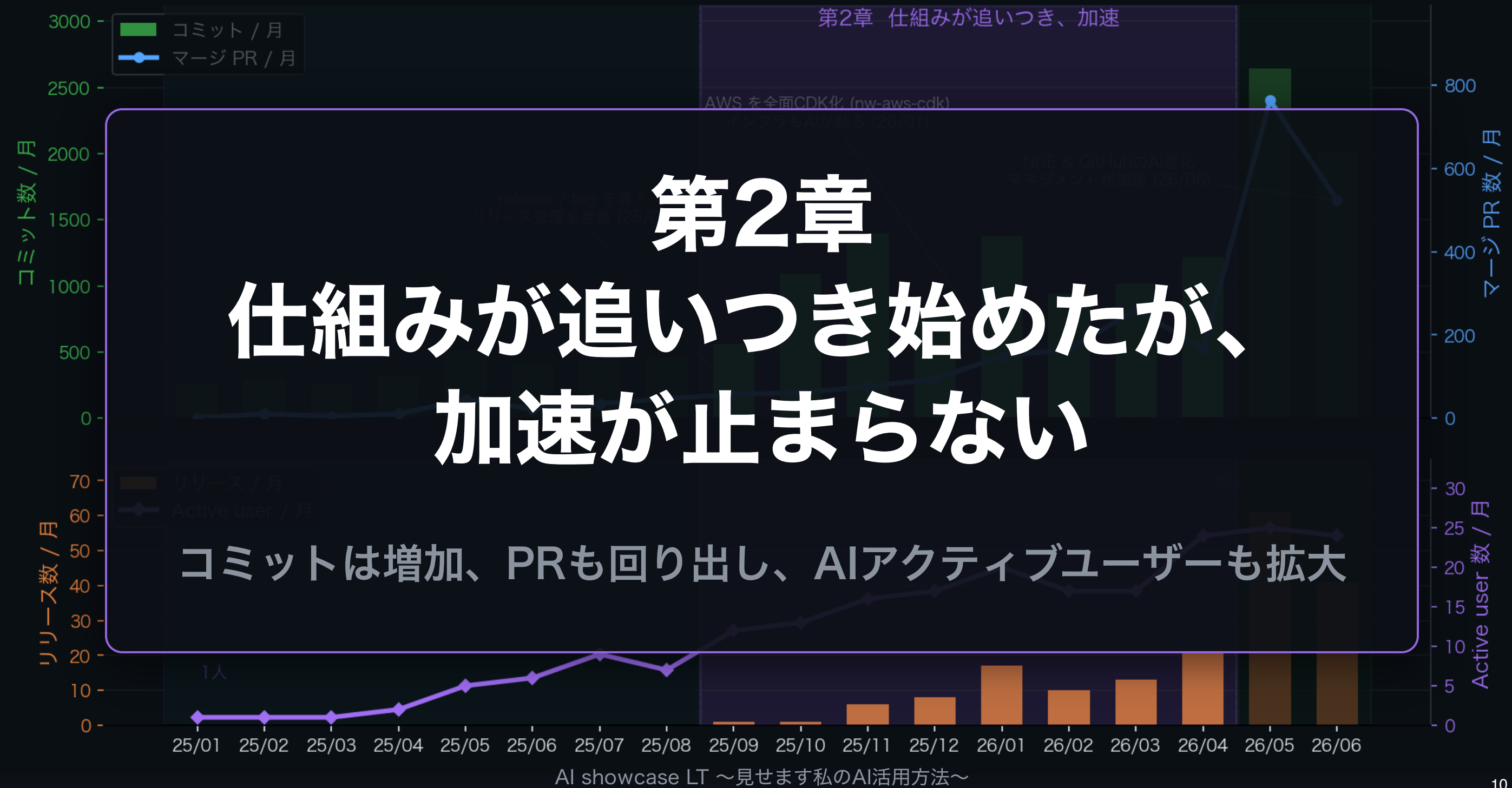
動くけど変なコード

コメントしては差し戻し
の繰り返し

 ボトルネック

コードの量産はできても、品質のコントロールができない
Modelの性能問題もあるが、コーディング経験不足が主因

→ 「品質をコントロールして開発を回す仕組み」が必要になった



第一歩：開発ライフサイクルの意識づけ

— 2025年 — 量産されたコードは何を作っているかを意識させる

「機能追加なのかバグ修正なのか」を曖昧にしない

- 意識する — Semantic Versioning と Releases で開発サイクルを意識させる
- 可視化する — リリースノートに「何が・なぜ変わったか」を残す
- 体験する — とにかく作ったものにverをつけてリリースしてみる

💡 体験から得る

不格好ではあったが、開発のライフサイクルを導入できた

→ 作っているものの解像度があがった。AIに説明できるのでコード品質があがったはず

回復：少しずつPRが回り出した

— 2025年9月～2026年4月（8ヶ月） — release/tagの仕組みを入れた後

書く量はさらに増えた

8,371

コミット
(月平均 ≈ 1,046)

**≈ 2.7
倍**

月コミット数
初期と比べて

だけど

なんとなく回っている

1,030

マージされたPR
= コミットの12.3%

6.5 → 12.3%

初期と比べてPR/コミット比がほぼ倍増
(月次は最大26%まで上昇)

ターニングポイント

コード量を加速させたまま、PRが“マージされる”ようになった

→ 小さい単位でPRを回す“ハーネス”が、AIの暴走を抑えた

加速：インフラ(AWS)もAIの管理下に

— AWS導入 + 管理を全てCDKで実施 (cdk-full-managed)

2026年1月、AWSを導入。

AWS経験者もいないので、最初から AI の管理下へ

これまで

オンプレIaaS
OS 払い出し・構築が手作業

いま

必要なリソースをAIが定義
CDKで自動作成・管理

⚡ 加速

**AWSをAIに管理させたことは大成功。
全てIaC化された恩恵がとても大きい**

→ インフラもAI の管理下に入り、Deploy と Release の障壁が一気に下がった

言語の壁が、無くなり始める

— リポジトリの言語別コード量（実データ・GitHub API集計）

学習コストはAIが肩代わり、言語は”性能・機能”で選べる



📍 選び方が変わった

言語選定の基準が「書きやすさ」から「適材適所」へ
並行してコンテナからServerlessへ、実行環境も変化

“作る”が解決したら、次が詰まった

— AIの圧倒的な生産スピードに、人間の管理プロセスが追いつかない

Modelの進化やAgent Codingの発展でさらにコード量は増え続ける

AIがコードを量産するほど、新たな課題が発生する



レビュー

PRが滞留して回らない



ブランチ戦略

シンプルな戦略だと
開発速度が速すぎて
mainがバーストする



リリース判定 Go / No-Go

誰が・何を基準に決める？
コードはあるが
運用変更は追いつくか？

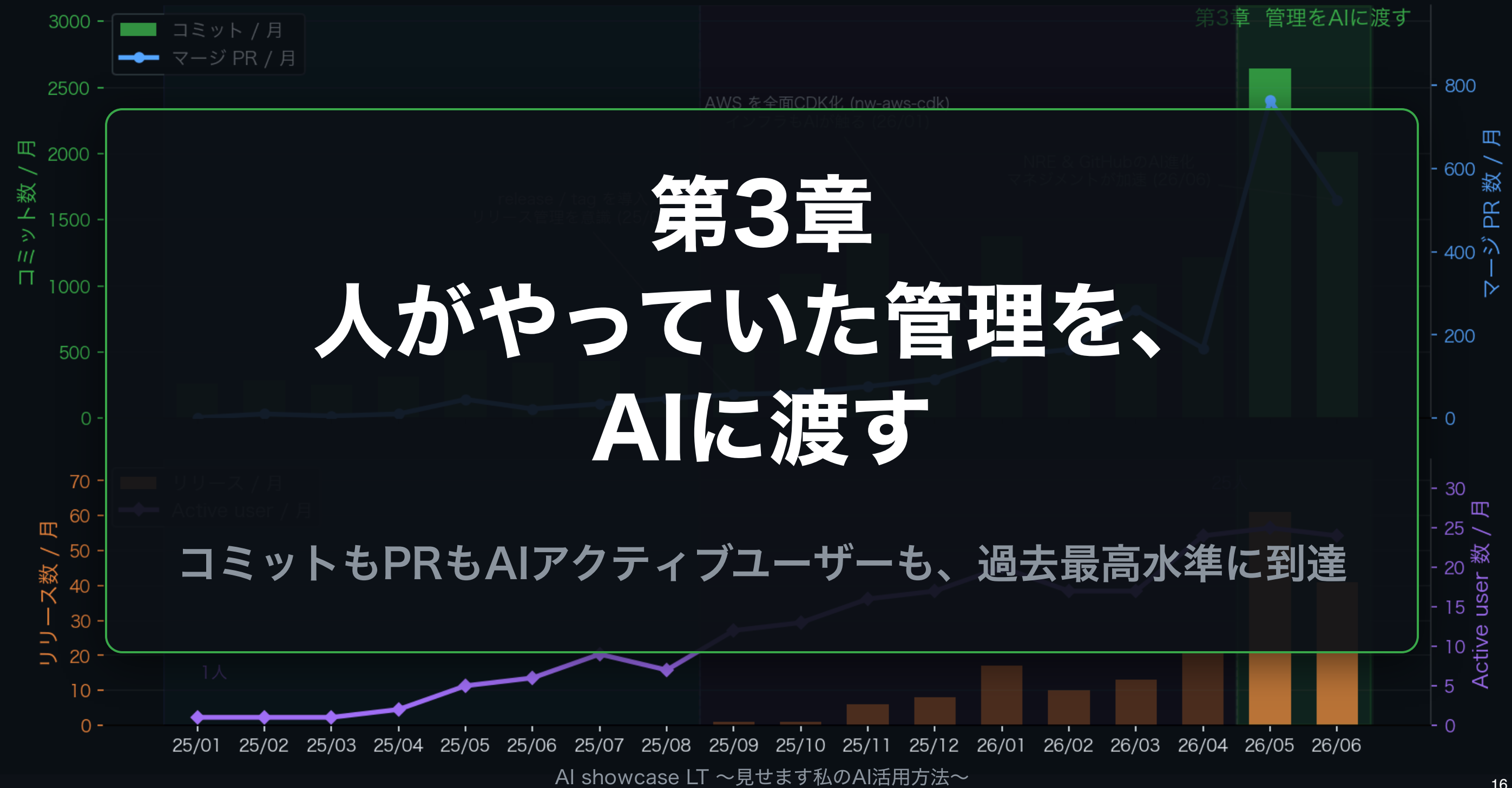


Issue / スケジュール

全体像が誰にも見えない
進捗が把握できない

🔄 ボトルネックの移動

ボトルネックが「作る」から「回す」に変わった



“回す”も、結局AIにやらせればいい

— コーディングも、インフラ管理も — 結局ぜんぶ AIにやらせてきた

コーディング → AIにやらせた **DONE**

インフラ管理 (AWS/CDK) → AIにやらせた **DONE**

ガバナンス / PM → AIにやらせる **同じはず**

人間ができるのであればAIにもできる。ガバナンスもPMも、AIに委ねればいい

「コードもインフラもAIにやらせたらよくなった」 — 答えは、ずっと同じだった

委ねる① プロジェクト管理をAIで

— GitHub Projects × Copilot — 複数リポジトリを跨ぐ進捗管理

「Taskを切って欲しい」と伝える、AIに委ねるプロジェクト管理

BEFORE — 人手

GUIをポチポチ手作業。
それを毎日くり返す
ルールを覚える必要があるし、
ルールは崩れる。



AFTER — AIが支援

Skillを定義すれば、
詳細なルールを覚える必要がない。
Release→**Epic**→**Task**を階層化。
Assignee Iteration の振り分けルール

 PMのAI化

決められたルールでプロジェクト管理を、AIが支援する

委ねる② ブランチ戦略もAIで

— AI Agentic Workflow — 開発者のブランチ戦略実行を支援する

「この改修、main にmergeしていい？」にAIが回答する

課題 ブランチ戦略が複雑になった — Branch作成やMerge先の判断が難しい

解決策 差分内容 × issueの背景から main直 / release経由 を判断支援。

 ブランチ戦略のAI化

曖昧なブランチ戦略・Mergeの
妥当性判断を、AIとSkillで固める

委ねる③ リリース判定もAIで

— PR Template × チェックリスト — リリース可否を「型」で揃える

「これ、リリースして大丈夫？」を**テンプレート+AIで標準化**

課題

観点が均一化されていない — 影響範囲・テスト・切り戻しの確認が曖昧

解決策

PR Template に Go/No-Go チェック項目を定義
AIが設計とコードから生成。CIと合わせリリースゲートを作る

 **リリースのAI化**

リリースの「OK」を、**人の勘から テンプレ+AI へ**

いちばん伝えたいこと

— 書けなかった → コードの99%をAIが書く → SDLC のガバナンスまでAIへ

**「AIに書かせる」で終わらせない。
開発ライフサイクル(SDLC)ごと、AIに委ねられる。**

極論、自然言語で表現できるものは全てAIでできる。
人がやるのは、“何を・なぜ世に出すか”を決めることと、AIに仕事を渡すことだけ。

Thank you

ご清聴ありがとうございました