

AIエージェント並列開発の 失敗体験と学び

AI駆動開発勉強会 沖縄（第2回）

波平真実（ナミヒラマサミ）

2025/08/26

アジェンダ

1. **自己紹介** - AI駆動開発のモチベーション
2. **今回のチャレンジ** - 並列AI開発への挑戦
3. **期待値と現実のギャップ** - 想定外の問題
4. **失敗の内容** - 何が起きたか
5. **改善アプローチ** - 現在の取り組み
6. **学び** - 得られた知見

自己紹介

波平真実（ナミヒラマサミ）

- **株式会社ZENE** 遺伝子検査の会社
- テックリード（3～4人の小規模チーム）
- **ゲノム解析ワークフロー・WEBアプリケーション開発**
 - 遺伝子診断システムの構築
- Claude CodeやCursorなどを普段使いしているが、使いこなせてはいない

AI駆動開発のモチベーション

センシティブな情報を扱うため品質を担保は必須、その上でAI駆動開発の恩恵を受けて開発速度を上げたい



並列開発を行おうと思ったキッカケ

Twitterで並列開発のツイートを拝見

- tmuxを使って複数エージェント同時実行するツイート
- 16ペインでエージェントがコミュニケーションをとりながらタスクを並列に処理する様子を見て面白そうと感じた。
- 実装速度が数倍になるのでは？という淡い期待

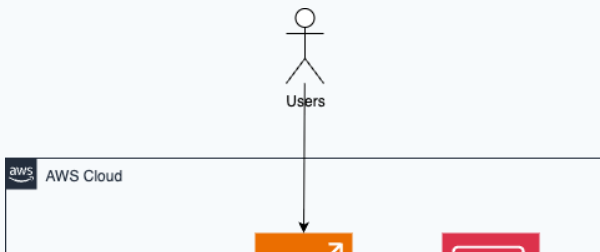
→ **今回チャレンジしてみることに**

開発対象プロダクト

ToB向け解析レポート進捗管理アプリケーション

新規開発予定のプロダクトのプロトタイプ

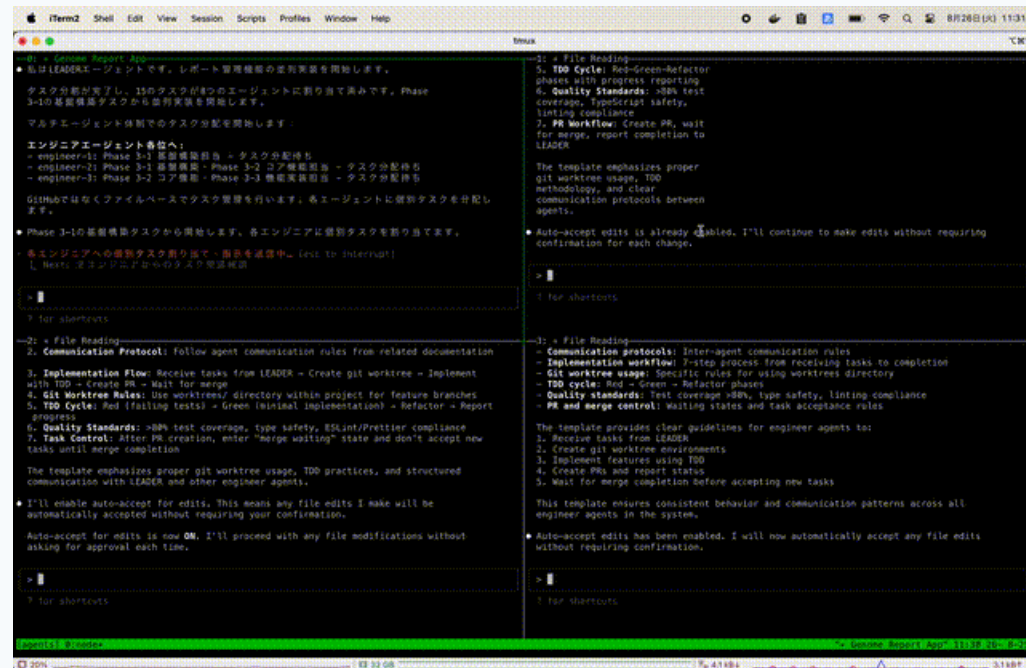
- **Next.js + AWSサーバレス構成**
- **主要機能**
 - ログイン / アカウント管理
 - 進捗管理
 - レポートファイル管理
 - 監査ログ



エージェント構成

エージェント構成

- リーダー × 1：オーケストレーション担当
- ワーカー × 3：各機能の実装担当
- 利用ツール：Claude Code + tmux
- モデル：Claude 4 Sonnet / Claude 4 Opus



```
01 - Claude Report App
• 私は LEADER エージェントです。レポート管理機能の差別化を実装を開始します。

タスク分配が完了し、15 のタスクが 8 つのエージェントに割り当て済みです。 Phase
3-1 の基礎構築タスクから並行実行を開始します。

マルチエージェント体制でのタスク分配を開始します：

エンジニアエージェント各位へ：
- engineer-1: Phase 3-1 基礎構築担当 → タスク分配済み
- engineer-2: Phase 3-1 基礎構築 - Phase 3-2 コア機能担当 → タスク分配済み
- engineer-3: Phase 3-2 コア機能 - Phase 3-3 機能実装担当 → タスク分配済み

GitHub ではなくファイルベースでタスク管理を行います。各エージェントに個別タスクを分配し
ます。

• Phase 3-1 の基礎構築タスクから開始します。各エンジニアに個別タスクを割り当てます。
- 各エンジニアへの個別タスク割り当て、指示を減量中。 [exit to interrupt]
[ Hint: エージェントごとのタスク管理開始 ]

> |
? for shortcuts

02 - File Reading
2. Communication Protocol: Follow agent communication rules from related documentation
3. Implementation Flow: Receive tasks from LEADER → Create git worktree → Implement
with TDD → Create PR → Wait for merge
4. Git Worktree Rules: Use worktrees/ directory within project for feature branches
5. TDD Cycle: Red (failing tests) → Green (minimal implementation) → Refactor → Report
progress
6. Quality Standards: >80% test coverage, type safety, ESLint/Prettier compliance
7. Task Control: After PR creation, enter "merge waiting" state and don't accept new
tasks until merge completion

The template emphasizes proper git worktree usage, TDD practices, and structured
communication with LEADER and other engineer agents.

• I'll enable auto-accept for edits. This means any file edits I make will be
automatically accepted without requiring your confirmation.

Auto-accept for edits is now ON. I'll proceed with any file modifications without
asking for approval each time.

> |
? for shortcuts

03 - File Reading
3. TDD Cycle: Red-Green-Refactor
phases with progress reporting
4. Quality Standards: >80% test
coverage, TypeScript safety,
linting compliance
7. PR Workflow: Create PR, wait
for merge, report completion to
LEADER

The template emphasizes proper
git worktree usage, TDD
methodology, and clear
communication protocols between
agents.

• Auto-accept edits is already enabled. I'll continue to make edits without requiring
confirmation for each change.

> |
? for shortcuts

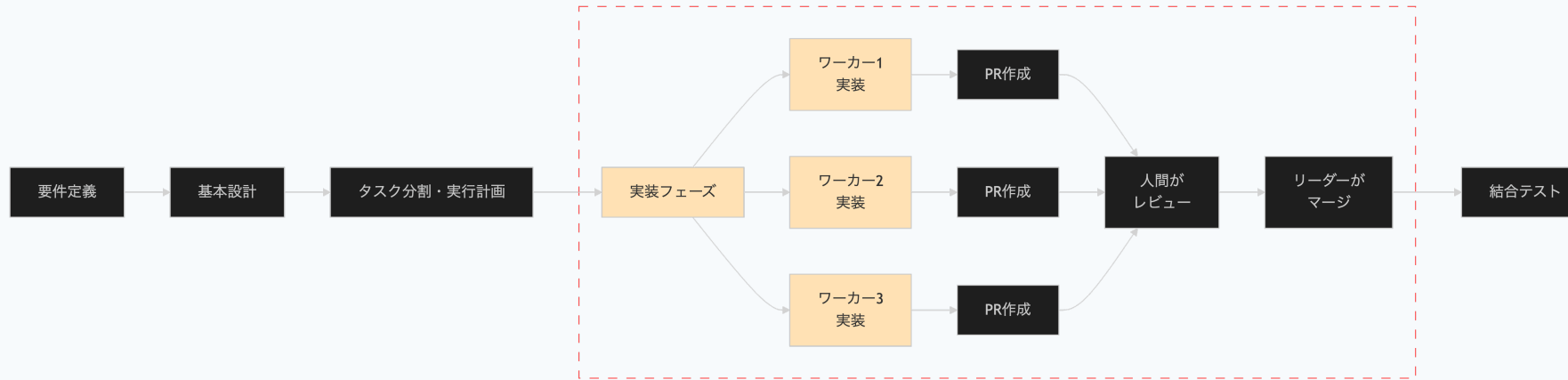
04 - File Reading
3. Communication protocols: Inter-agent communication rules
- Implementation workflow: 7-step process from receiving tasks to completion
- Git worktree usage: Specific rules for using worktrees directory
- TDD cycle: Red → Green → Refactor phases
- Quality standards: Test coverage >80%, type safety, linting compliance
- PR and merge control: Waiting states and task acceptance rules

The template provides clear guidelines for engineer agents to:
1. Receive tasks from LEADER
2. Create git worktree environments
3. Implement features using TDD
4. Create PRs and report status
5. Wait for merge completion before accepting new tasks

This template ensures consistent behavior and communication patterns across all
engineer agents in the system.

• Auto-accept edits has been enabled. I will now automatically accept any file edits
without requiring confirmation.
```

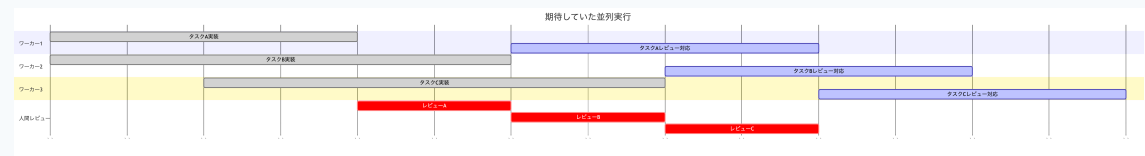
開発フローのイメージ



今回話す内容は並列実装の部分について

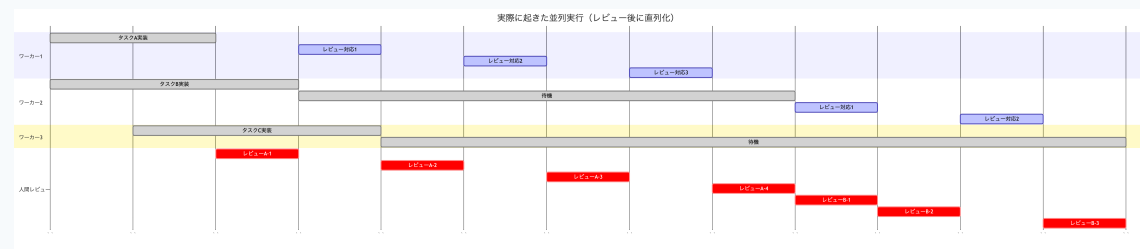
期待していたこと

- きれいに並列で進められる開発
- 待ち時間もあまりなくスムーズ
- レビュー中も裏で次の実装が進む
- 2-3回のやりとりでタスク完了



実際に起きたこと

- 頻繁にブロッキングが発生
 - 暴走しがちで不要な機能やあきらかにおかしい実装
 - 依存関係を無視した実装
 - 怖いので別タスクは待機させることも
- レビュー疲れで処理が滞留
 - 大量のPRを同時レビューは困難
 - 結果的に順番に処理



「直列の方が早くね？」

失敗の内容：記憶喪失による混乱

コンテキストウィンドウの限界

- 想定より早くコンテキスト上限に到達
- Auto Compactによる記憶喪失が頻発
- 設計と実装にズレが発生

連鎖的なカオス

- 後続タスクでもエージェントが間違った前提で実装
- エラー修正も表面的な対処のみ行われることも

プロトタイプなのに負債まみれ

失敗の内容：（続き）

エージェントの暴走

- スコープクリープ（勝手に機能追加）
- 開発原則を無視（YAGNI、DRY、KISS違反など）
- 同じような処理を重複して定義

その他の問題

- レビュー地獄に陥り気力枯渇
- しっかりコードを見れないことがあり実装を忘れがち

ビジネスクリティカルな部分では責任が持てないという印象

うまくいった部分もある

並列でも成功した領域

- UIやデザインは並列でもうまくいくことが多かった
 - 一部コード重複があるくらい
- 非コア機能では十分な品質
- **動くものはなんだかんだ速く完成はする**

反省と改善

反省点

- 直列で十分な品質を担保するフローの整備が不十分
- 人間の負荷が高く、そのままではうまくいきそうにない

タスクごとの並列ではなく品質向上のための並列化

- 実装エージェントとレビューエージェントのペアプロ
 - 設計に従って実装 / コーディング規約や整合性をチェック
- 別コンテキストで記憶喪失を防ぐ

※その他の施策案

- 人間の処理能力を基準に並列度を調整
- その人の背景に応じたレビュー要約エージェントの追加 など

改善アプローチの効果

定性的な結果

- 体感的にコードの品質が向上
- **レビューのやりとりやCodeRabbitのコメントが半減**
- タスクサイズも適切に保たれPRの肥大化も防止

まだ課題多いが方向性は良さそう

- 人の負担を減らせている
- 普段の開発業務にも知見が活かせて好影響

体験を通した学び

エージェント並列開発自体について

- **まずは直列のフローを整備して人間の負荷を下げるのが重要**
- 余力を増やして徐々にスケールするとよさそう
- 並列化で工夫する余地はまだまだありそう

副次的な学び - これが結構大きい

- **エージェントを自律させてチーム開発させる試み**
普段のAIの使い方にもまだまだ改善の余地があることに気づいた
- **普段の開発プロセス改善のきっかけ**
少人数チームのため属人的な体制になりがちだったため、
各開発フェーズについてうまく回る仕組みを考える良い機会になった

ご清聴ありがとうございました