Code Navi

Slack連携型 AIマルチエージェントシステム

- Slack連携
- コード分析
- ST作成
- PRレビュー

不便を自分で解消する


ST資料作成
レビュー
影響範囲調査
UT作成
チケットの内容確認
コードリーディング
エラー調査
設計書作成
リリースノート作成
ドキュメント作成
テストデータ作成
見積もり
環境構築
マイグレーション作成
API仕様書作成
議事録作成

// Slack連携型 AIマルチエージェント

普段の会話の延長線上で使える。
チケットを起点に、コード分析・レビュー・ST作成を一貫して対応。

// 何ができるか

▪ Issue管理

Backlogチケットとの紐付け。
チケット情報を起点に作業を開始。

▪ レビュー観点作成

機能・非機能・品質の観点を自動生成。
レビューの基盤になる。

▪ PRレビュー

観点に基づくコードレビュー。
結果はSlackに報告。

▪ ST作成

影響分析→観点→パターン→Excel。
結合テスト仕様書を自動生成。

▪ コード分析

シンボル検索・参照追跡。
LSPベースの正確な解析。

▪ メモリ

過去の会話・判断を引き継ぐ。
文脈を跨いだ対話が可能。

実際の成果物

ST資料

結合テスト仕様書
影響分析→観点→パターン→Excel

PRレビュー

観点に基づくレビュー結果
Slackに自動投稿

SECTION 02

Code Naviの設計

AIの成果物の質は、何を渡すかで決まる。
3層のコンテキスト管理で情報を絞る。

// 前作の失敗

PRの差分だけを見る
レビューエージェント

関数の引数を変更した。
差分にはその変更だけが映る。
呼び出し元への影響は見えない。

- ✓ 関数の引数を変更
  ↓ 差分にはこれだけ
- □ 呼び出し元A – 差分に出ない
- □ 呼び出し元B – 差分に出ない
- □ 呼び出し元C – 差分に出ない

- **コンテキストが足りなかった**

AIの成果物の質は「何を渡すか」で決まる

- ~~全部渡す~~
- ~~差分だけ渡す~~
- ✓ 必要な情報を、必要なだけ渡す

// 情報は足りなくても、多すぎてもダメ

■ 不足

情報が足りない

PRの差分だけ渡す
→ 呼び出し元の影響が見えない

- 推測に頼る → 精度低下

■ 過多

情報が多すぎる

プロジェクト全体のコードを渡す
→ ノイズで判断が混乱

- 情報が混ざって混乱

■ バイアス

視点が偏る

実装したままのコンテキストでレビュー
→ 批判的な視点が出ない

- 批判的な視点が出ない

// 3層のコンテキスト管理

コンテキストを
絞る

それぞれ異なる場面で、
同じ原則を適用している。

01 マルチエージェント
指示を受けたとき
役割に応じたプロンプト・ツールに切り替え

02 サブエージェント
分析タスクを実行するとき
専門領域ごとに判断範囲を限定

03 Serena（LSP）
コードを読むとき
シンボル単位で必要な情報だけ取得

// 第1層：マルチエージェント

会話履歴は共有、
役割は切り替え

ユーザーの指示に応じて、専門エージェントに切り替わる。会話の文脈は維持したまま、プロンプトとツールだけが変わる。

- **General**
  汎用対話
- **StCreator**
  ST作成
- **PRReviewer**
  レビュー

$ ユーザー：「STを作って」

↓

- **GeneralAgent**

  「ST作成の仕事だ」

↓ handoff

- **StCreatorAgent**
  - ST専用のプロンプト
  - ST専用のツール
  - 会話の文脈は維持

// 第2層：サブエージェント

判断範囲を
専門領域に限定

1つのタスクを複数の専門家に分割。
各サブエージェントは自分の領域の情報だけを扱う。

ST影響分析

- **広域探索**
  影響範囲の特定
- **API影響分析**
  エンドポイント
- **DB影響分析**
  テーブル・カラム
- **画面影響分析**
  UIコンポーネント

PRレビュー

- **観点チェック**
  レビュー観点の検証
- **命名チェック**
  規約準拠
- **データフロー**
  入出力の整合性

// grep vs LSP

BEFORE

grep（テキスト検索）

handleSubmit を検索した場合

- **handleSubmit** = () => Form.tsx
- ~~// handleSubmit を呼ぶ~~ Form.tsx
- ~~handleSubmitData(...)~~ test.ts
- ~~"handleSubmit_label"~~ labels.ts

1/4 本当に必要な結果は
4件中1件だけ

vs

AFTER

Serena（LSP解析）

handleSubmit の参照元を取得した場合

- ✓ **FormPage.submit()** 呼び出し
- ✓ **FormValidator.validate()** 呼び出し
- ✓ **FormTest.test_submit()** テスト

3/3 全件が実際の参照元
ノイズなし

SECTION 03

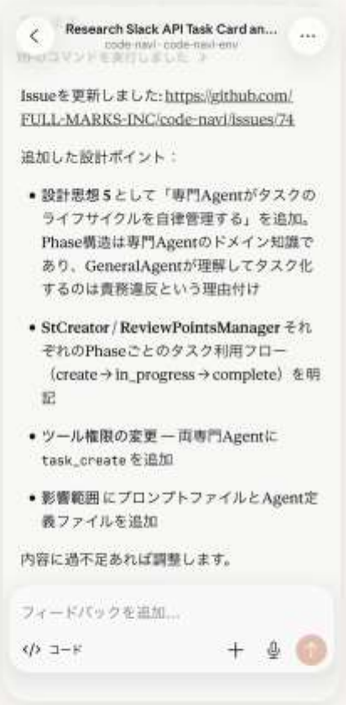
バイブコーディングの知見

AIと対話しながらコーディングを進める。
開発者は設計判断と方向修正に集中する。

// 実際の開発フロー

スマホでIssue作成

電車の中で調査・起票



Issueを実装させる

Issueを見せながらコーディング

// 品質との付き合い方

- ~~メソッド単位のレビュー——AIの速度に追いつかない~~

  AIが生成するコードを1行ずつ確認する。レビューがボトルネックになる。

- ✓ **ディレクトリ構成・クラス設計を握る**

  全体の方向性だけは自分で決める。

  構造が正しければ、細部はAIに任せられる。

// 先駆者から学ぶ + 最新情報をキャッチ

LEARN

車輪の再発明を避ける

自分が悩んでいることは、先人が既に解決していることが多い。

- 悩み：エージェントの粒度

↓

- 参考：Claude Codeのサブエージェント設計

↓

- 発展：自分のアイデアに昇華

CATCH UP

最新情報を取り入れる

開発中に、開始時にはなかった機能が次々登場。
取り入れることでシステムが大きく改善できた。

- Slackストリーミング
- PlanCard
- Strands SDK Swarm

// 挫折も糧になる

01

Zed EditorのLSP統合

Zed Editor向けに自作LSPを作ろうとした。
既存LSPのラップで良かったのに1から作ろうとして断念。

- アプローチの選択ミス

02

GitHub差分ブラウザ拡張

GitHubのPR画面を拡張するブラウザ拡張。
技術的には動いたが、作っていて楽しくない。
GitHub側のレイアウト変更に対応できない。

- 面白くなくて続かない

03

PR差分の自動レビュー

PRの差分だけを見るレビューツール。
コンテキストが足りず精度が出ない。

- コンテキスト不足で精度が出ない

この経験が、Code Naviの設計判断を支えた。

// 向いている場面

1人開発に向いている

設計思想を自分だけが持っていればいい。
AIとの対話で素早くイテレーションできる。

- ✓ **設計判断が速い**
  自分で決めてすぐ反映できる
- ✓ **方向転換が自由**
  ダメなら壊してやり直せる
- ✓ **隙間時間で進められる**
  通勤中でも開発が進む

- □ **複数人開発は課題あり**
  設計思想の共有、レビュー負荷、一貫性の維持

コードを書くのはAI
方向を決めるのは自分

自分で解消できる

ST資料作成
レビュー
影響範囲調査
UT作成
チケットの内容確認
コードリーディング
エラー調査
ドキュメント作成
設計書作成
テストデータ作成
リリースノート作成
議事録作成
環境構築
マイグレーション作成
見積もり
API仕様書作成

まず、使ってみてほしい

使ってみると、できないことがたくさんある。

テストデータのExcel作成、チケットの内容把握、AAAパターンでのUT作成、的外れな提案...

でも、それは解決しうること。

Code Navi

Slack連携型 AIマルチエージェントシステム

- Slack連携
- コード分析
- ST作成
- PRレビュー