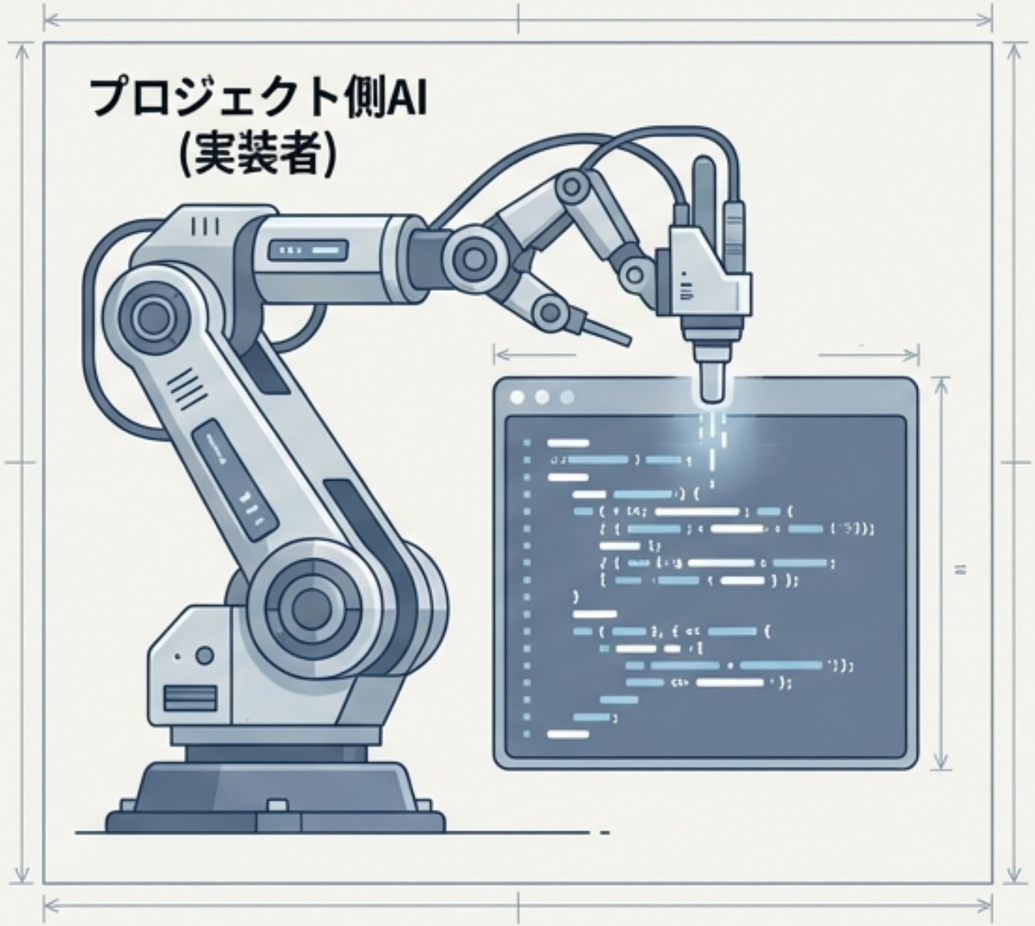
独立したAIによる第三者検証のパラダイム

Test Insight Hub: 継続的品質改善エンジンの導入と実践ガイド

プロジェクト側AIが書いたコードを、監査役AIが検証する。
品質担保のプロセスを自動化・可視化する6つのステップ。

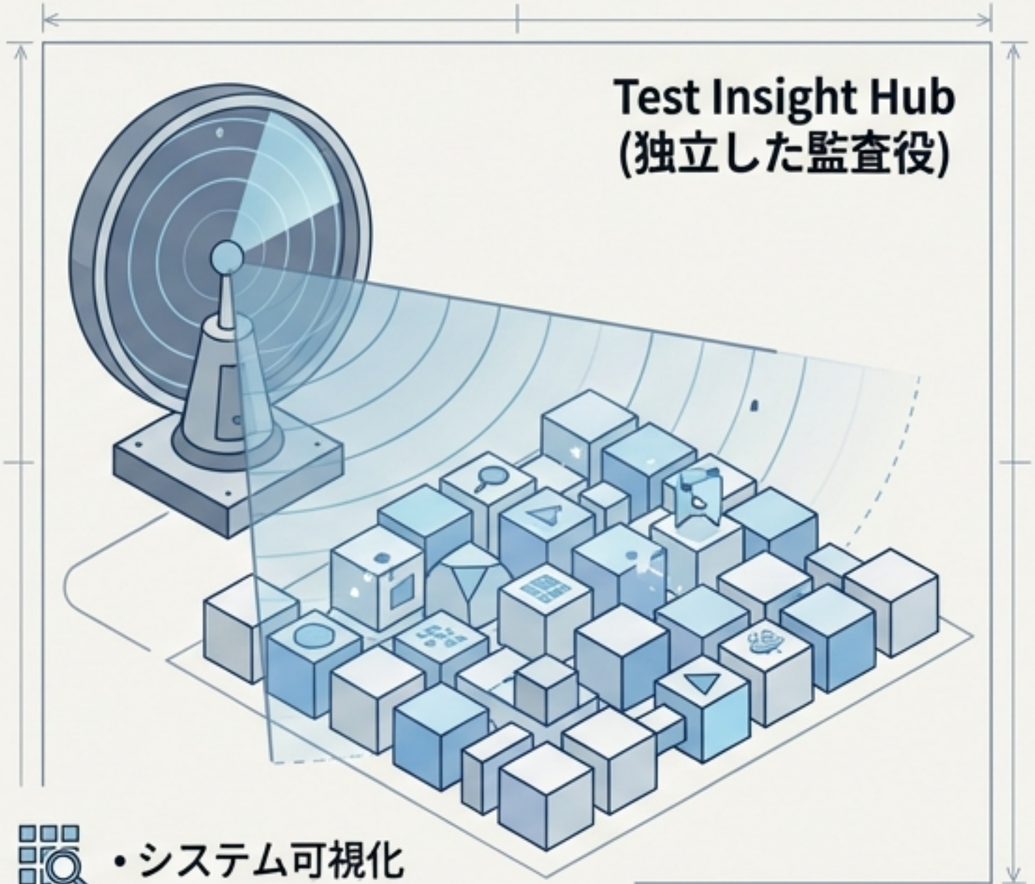
実装者AIと独立した監査役AIの分離

AI駆動開発における最大のブラックボックスはテスト品質です。Test Insight Hubは、実装を担当するプロジェクト側AIとは独立した監査役として機能し、システムの可視化、カバレッジ計測、改善フィードバックを自動化します。



コード生成、機能実装

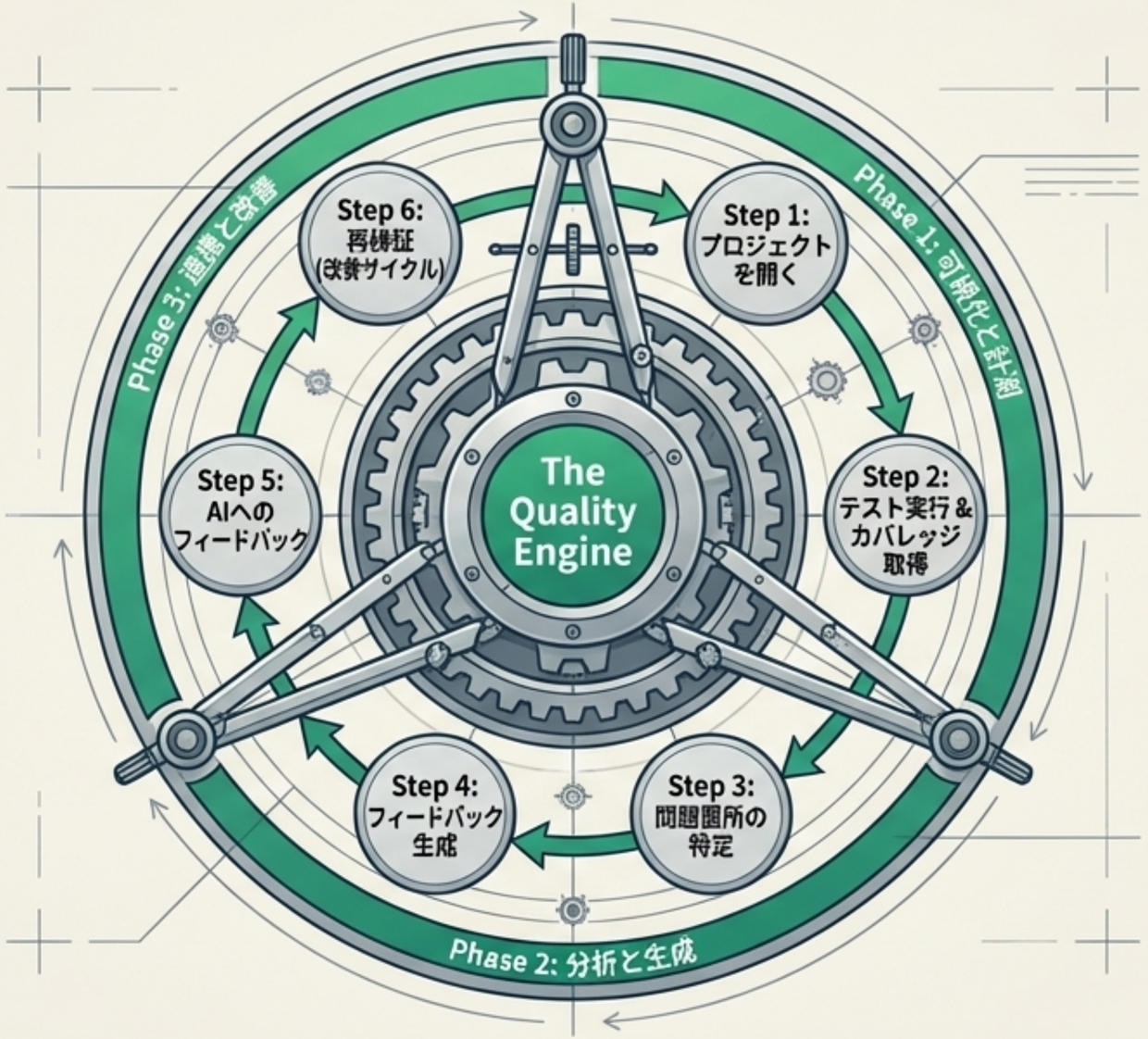
客観性の壁



- システム可視化
- カバレッジ計測
- 改善フィードバック自動化

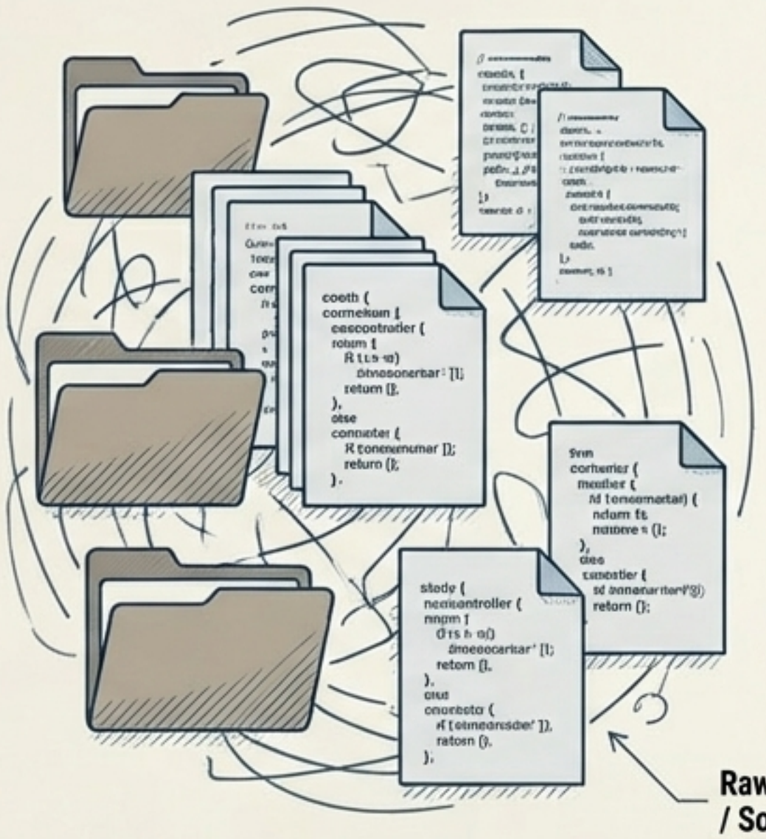
継続的品質改善サイクル (The 6-Step Engine)

これは直線的な手順ではありません。Quality Gateが合格になるまで、このエンジンを回し続けます。



Step 1: システム構造の自動解析と可視化

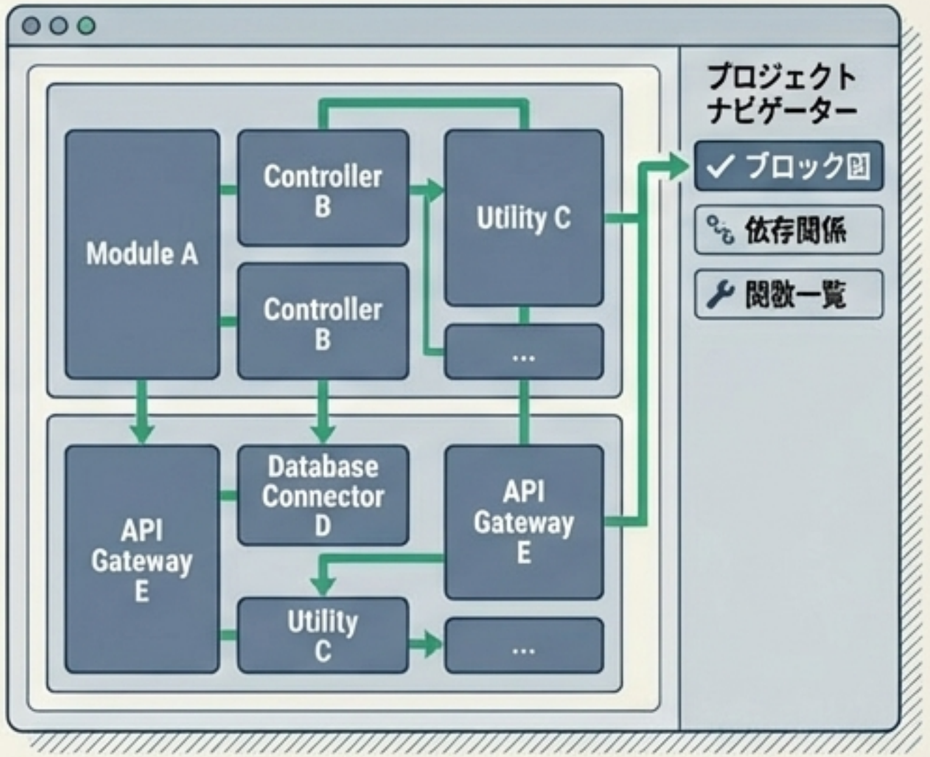
ホーム画面からテスト対象のプロジェクトディレクトリを選択するだけで、裏側の複雑なソースコード群が瞬時に解析されます。
抽出されたモジュール構造や依存関係は、サイドバーの「ブロック図」として直感的なUIにマッピングされます。



Raw Directory Structure / Source Code

自動解析（モジュール構造・依存関係・関数一覧）

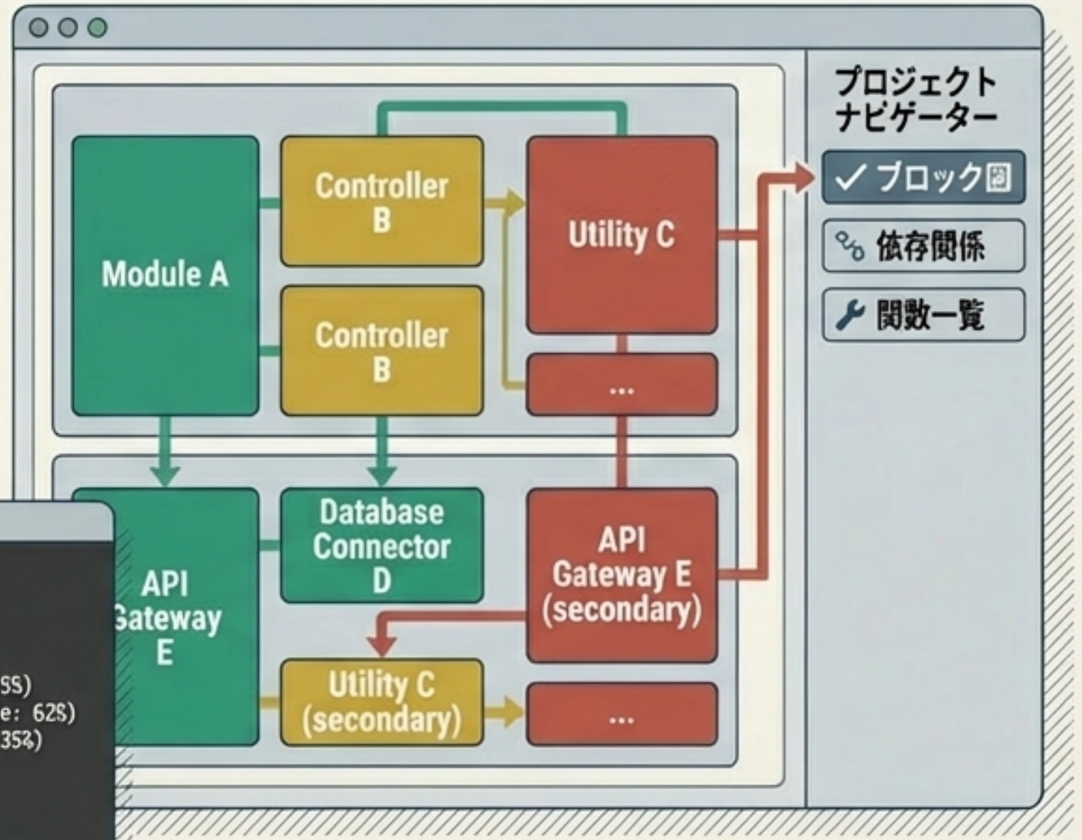
直感的なUIでのブロック図可視化



Step 2: テスト実行とヒートマップへの反映

フレームワークを自動検出し、テストを実行。完了と同時に、ファイル別の達成率がブロック図に
ヒートマッ図に色分けして反映され、システムの健全性が一目でわかります。

直感的なUIでのブロック図可視化 (ヒートマップ)



Legend

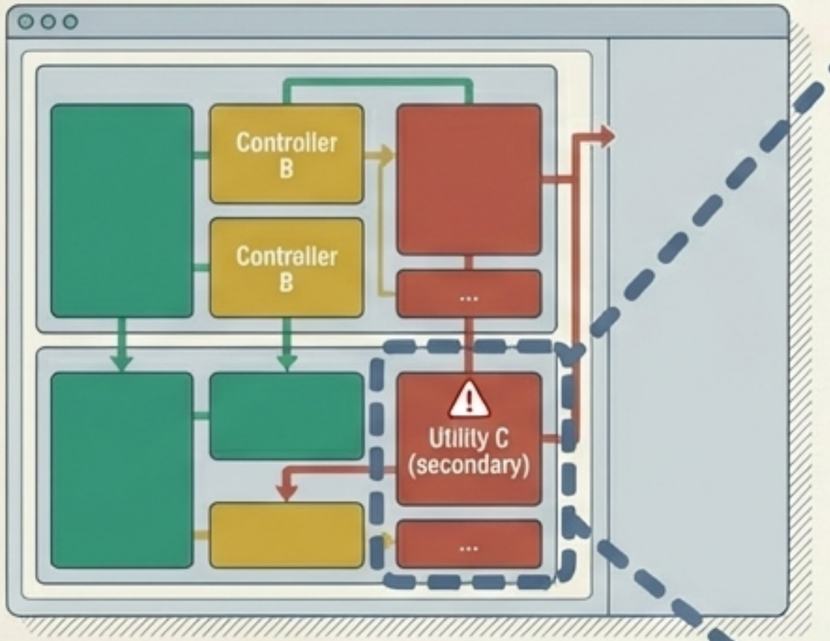
- カバレッジ 80%以上 (Safe)
- カバレッジ 50%以上 (Warning)
- カバレッジ 50%未満 (Critical)

リアルタイムログ

```
> test_runner --project=webapp
> Running tests...
> [INFO] Found 15 test suites
> [INFO] Executing tests...
> [SUCCESS] Module A tests passed (Coverage: 85%)
> [WARNING] Controller B tests passed (Coverage: 62%)
> [FAILURE] Utility C tests failed (Coverage: 35%)
> [INFO] Updating heatmap...
> [COMPLETE] All tests finished in 4.5s
```

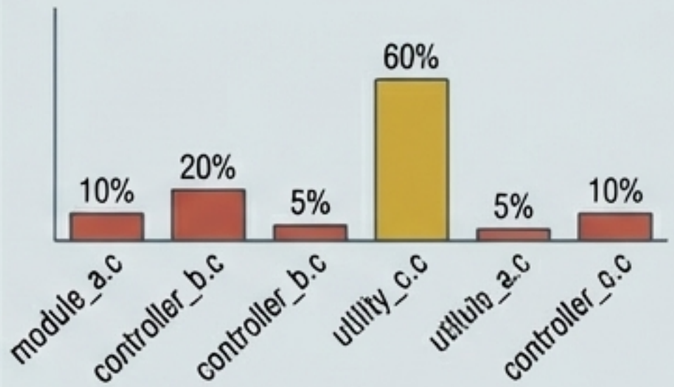
Step 3: 問題箇所の特定とドリルダウン

ブロック図上でアラートのモジュールをクリックし、詳細な課題を特定します。
カバレッジ不足の原因を確認し、直接フィードバック生成画面へ移行できます。



詳細分析パネル

ファイル別カバレッジ (%)



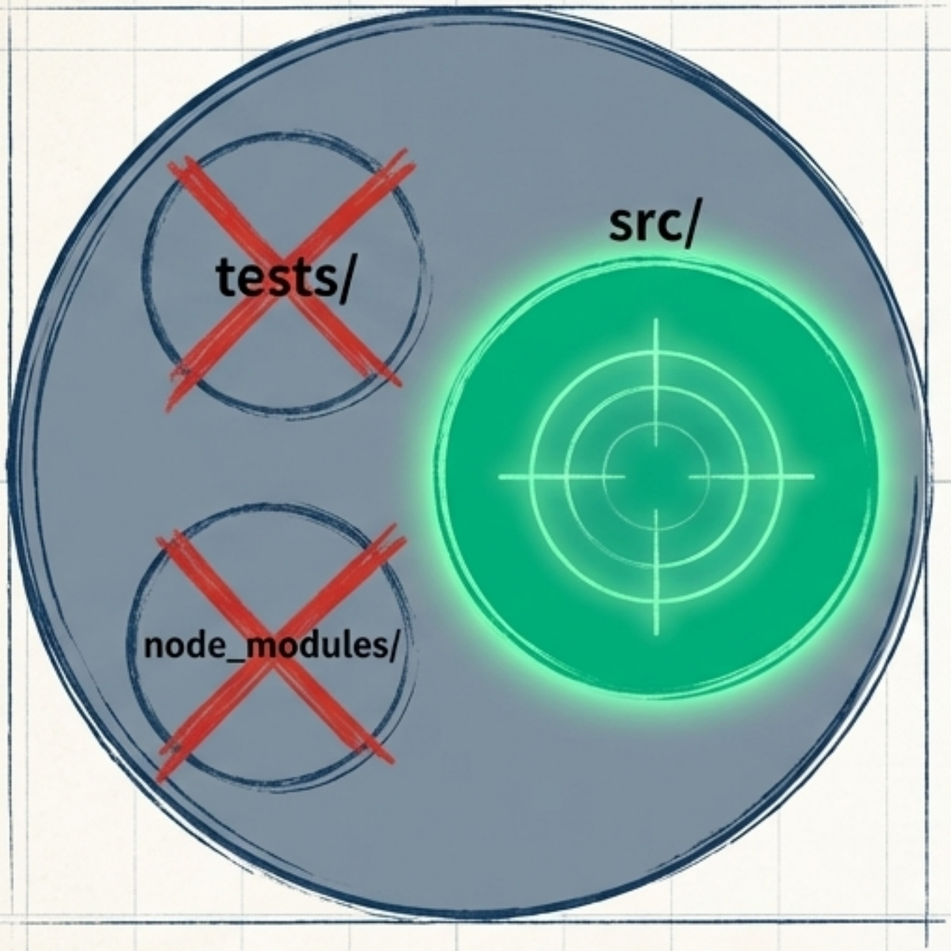
未テスト関数リスト

```
validate_input
process_data
handle_error
generate_report
```

対応テスト一覧

- [x] test_validation
- [ ] test_processing
- [ ] test_error_handling
- [x] test_report_gen

Step 4: スコープの限定と優先度の算出



フィードバックの対象は src/ 配下のソースコードに厳格に限定されます。 tests/ や node_modules/ は除外されます。

$$[\text{カバレッジ不足率}] \times [\text{コード複雑度}] \times [\text{変更頻度}] = \text{【優先度スコア】}$$

限られたリソースで最大の効果を出すため、3つの変数を掛け合わせ、最もリスクの高いファイルに優先度スコアを付与します。

The Quality Gate: 妥協なき品質判定

Mechanism: フィードバックには厳格なQuality Gateが含まれます。判定基準は全体の平均ではありません。
Strict Rule: 全てのモジュールが設定した閾値を超えて初めて合格となります。1つでも未達があれば不合格となり、差し戻されます。

Step 5: 実装側AIへのコンテキスト・ハンドオフ

生成されたシナリオをプロジェクト側AIに渡し、テスト改善を指示します。用途に応じて2つの手段を使い分けます。

	📋 コピー	📂 プロジェクトに配置 (feedback.json)
特徴	即座にチャットAIに貼り付け可能	.test-insight/feedback.json を生成
用途	特定の単一ファイルや小規模な改善時	プロジェクト全体の網羅的な改善時
AIへの指示	「この内容を元にテストを修正して」	「feedback.jsonを読み込み、Quality Gateが合格になるまでテストを追加して」

警告: Quality Gateが不合格の場合、特定モジュールだけを改善して終了せず、全モジュールが閾値を超えるまでAIに改善を継続させてください。

Step 6: 再検証とフライホイールの回転

プロジェクト側AIがテストを修正したら、再びStep 2へ戻りカバレッジを取得します。
フィードバック履歴画面で前回との比較・改善率を確認。全モジュールが基準を満たしているか再確認します。
Quality Gateが合格になるまで、このループを回し続けます。

統合エコシステムと対応フォーマット

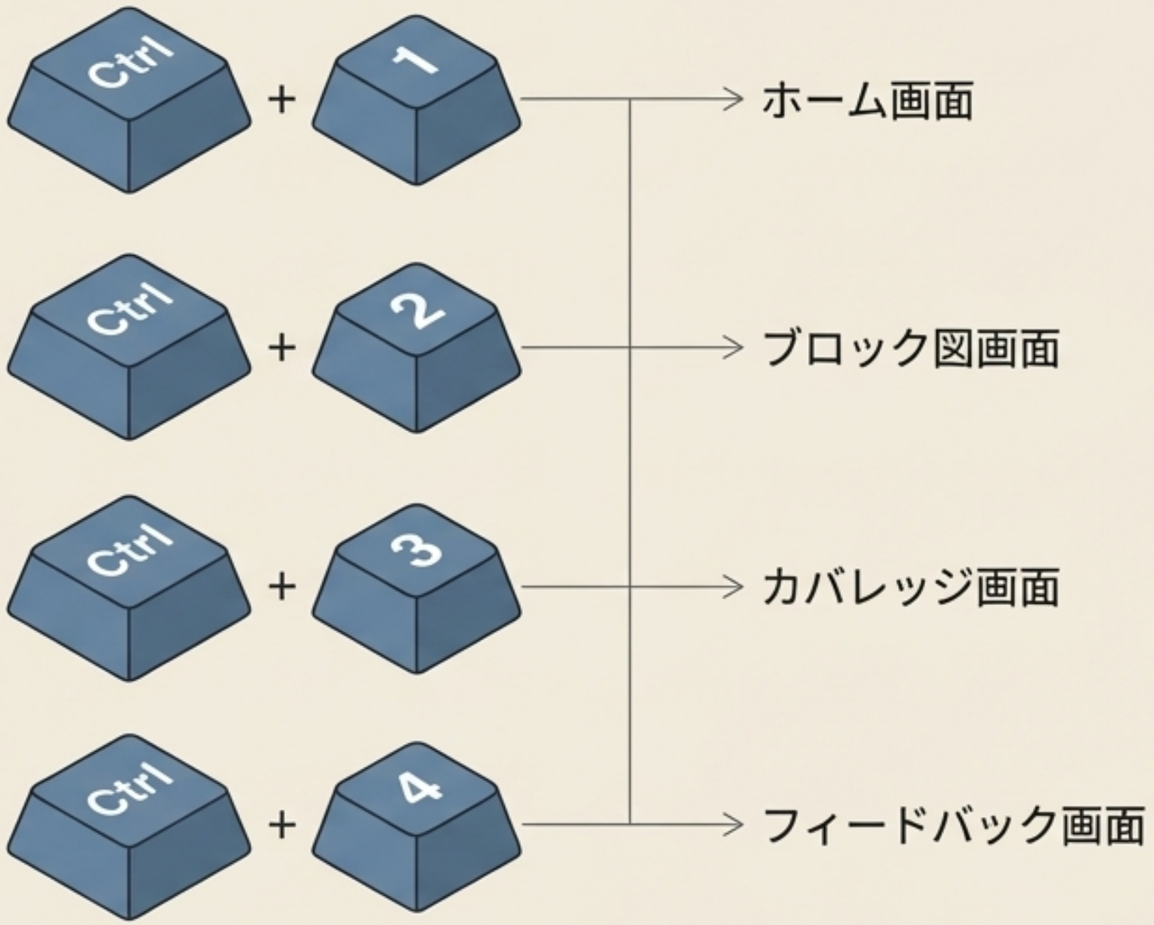
- TypeScript / JavaScript
- Python
- Go
- Rust
- C / C++

対応カバレッジ形式 × ツール

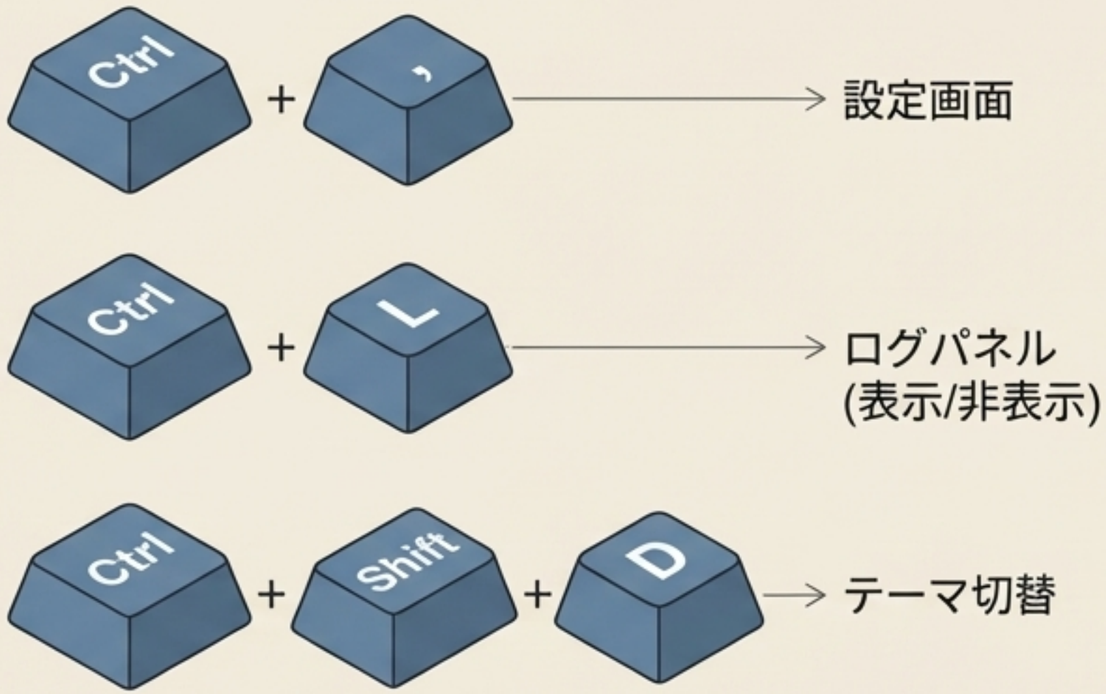
カバレッジ形式 (Format)		対応ツール (Tool)
Istanbul JSON	→	Vitest, Jest
lcov.info	→	汎用
coverage.py JSON	→	pytest
Go coverprofile	→	go test
llvm-cov JSON	→	cargo-llvm-cov

The Command Center (キーボードショートカット)

Navigation (ナビゲーション)



Utilities (ユーティリティ)



人間はコードを書くのではなく、AI同士の品質改善ループを指揮する

Synthesis Insight: Test Insight Hubの真の価値は単なるカバレッジ計測ではありません。「実装するAI」と「監査するAI」による自律的な品質担保ループを構築することにあります。

Closing thought: 開発者の役割は、この終わりのない改善サイクルのオーケストレーターへと進化します。システムを可視化し、高い品質水準を保ち続けてください。