MORE ADVANCED

SPIJapan2025

生成AIを活用したテストコード自動生成

～テストの未来を変える!?　テストコード自動生成への挑戦!　～

2025年10月23日

パナソニック アドバンストテクノロジー株式会社
森下直人　・　中前直義

はじめに

ソフトウェアは大規模化、複雑化しており、開発現場の負荷は常に高い状況です

我々、自動化推進Gでは開発現場の自動化を推進してきましたが
単体テストコードの整備は人手であり時間がかかる工程でした

そこで、生成AIを活用したら効率化できるのではないかと考えました

今回、生成AIを活用した単体テストコードの自動生成にTryしましたので
その内容や現場での活用実績についてご報告します

参考になれば幸いです

本日の内容

■ 会社紹介

■ 自己紹介

1. 背景／改善したいこと
2. 生成AI活用の試行錯誤
3. テストコード生成の効率化
4. 生成AIの実践活用と結果
5. 今後の予定

会社紹介

MORE ADVANCED

パナソニック アドバンストテクノロジー株式会社
Panasonic Advanced Technology Development Co., Ltd

設　立	2007年4月1日　(創業1985年)
事業目的	システムおよびソフトウェア設計開発を通じて 安全・安心、快適・便利な暮らしを実現する
従業員	519名　(2025/4/1時点)

① 受託開発が主ビジネス

パナソニックG

Panasonic

グループ外企業

② 技術者比率が高い

技術者比率
90％

③ 複数拠点での開発

自己紹介

MORE ADVANCED

1988 1990 1995 2000 2005 2010 2015 2020 2025

▲松下ソフトリサーチ入社

▲パナソニックアドバンストテクノロジー発足
（三社合併による社名変更）

ソフトウェア開発
［半導体設計・情報システム］

弊社内活動

SPI活動
ISO9000
CMMI

SEPGJapan2004 ▲
メンタルマネジメントを利用した現場密着型
SQA活動による効果的なプロセス改善事例

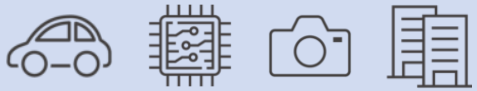
解析・自動化推進
生成AI

対外発表 ▲
Understandを用いた構造的
リスクの効果的な抽出手法とは

SPIJapan2025 ▲

**パナソニックHD
事業会社向け活動**

SPI・品質強化活動



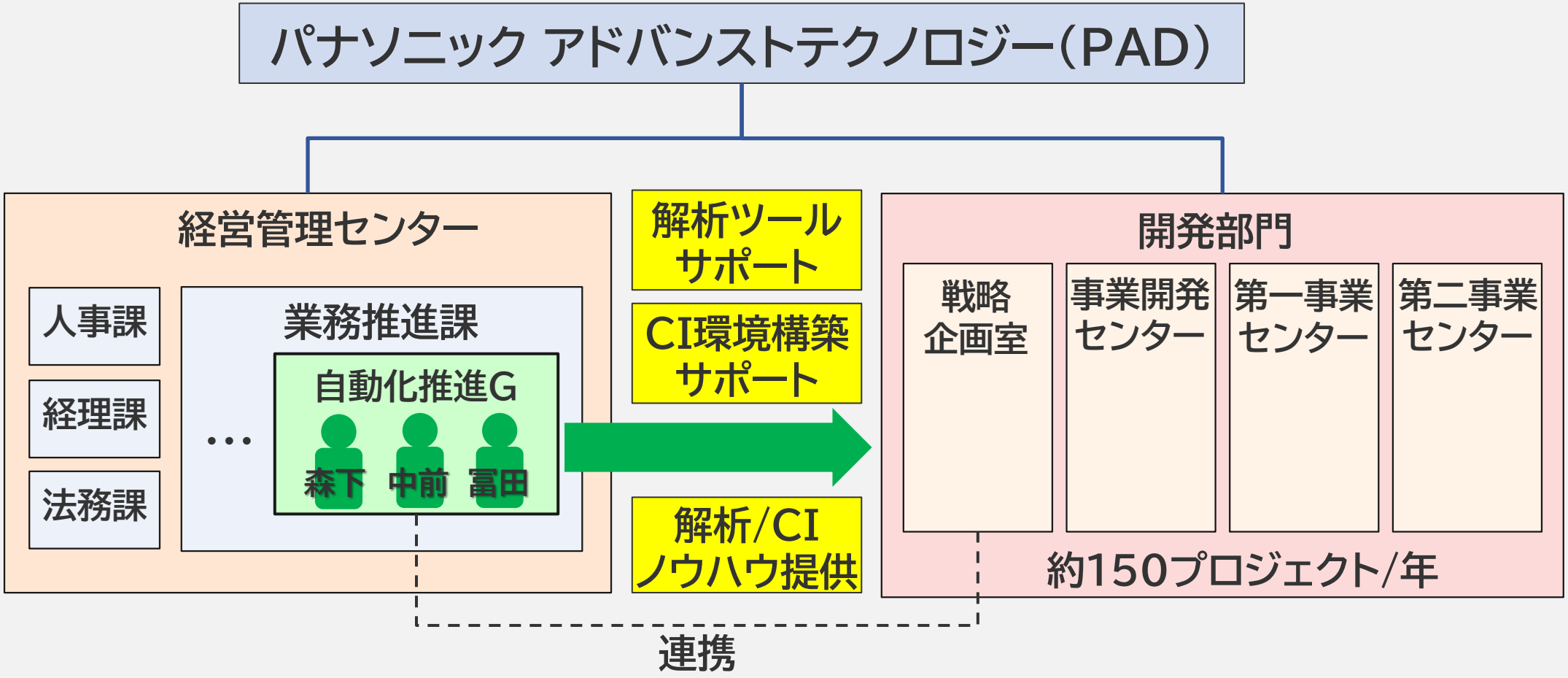
同左

SPI・PMO
プロセス講座

（連名）SPIJapan2013 ▲
テスト設計プロセス可視化の取り組み

1．背景・改善したいこと

業務推進課　自動化推進Gの役割



・静的解析/構造解析ツール、CI環境構築のサポート、及び、
ノウハウ化と発信により、開発部門の品質/生産性向上を推進

これまでの取り組み

MORE ADVANCED

2017 2018 2019 2020 2021 2022 2023 2024 2025

静的解析ツール

CODESonar CODESECURE ＋ Jenkins

品質問題（2件）

静的解析ツール 導入・自動化・サポート

ツール評価・選定

▲解析自動化

▲警告集計ツール発信

毎回手動だと手間かかるな

Jenkins導入

不具合分析

構造解析ツール

Understand™ by SciTools

複雑なコードはバグが多いな

SW構造解析のサポート開始

SW構造解析ツール　導入・サポート

▲ SW構造解析ガイドライン発信

ソースコードから詳細設計書が自動生成可能に

▲詳細設計書自動生成ツール整備・発信

CI環境構築

Jenkins ＋ googletest 他

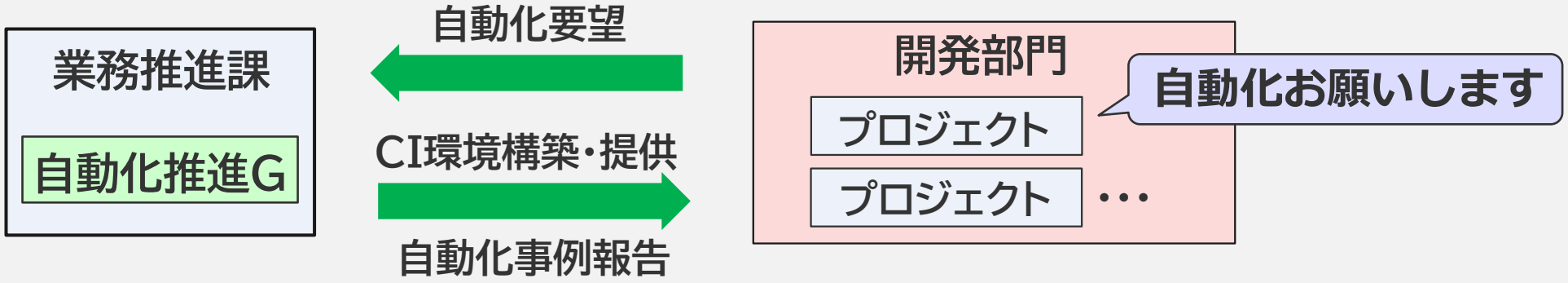
CI環境構築サポート

CI環境構築手順書発信 ▲

テスト実行も自動化

単体テスト自動化支援

CI環境構築支援



2025/9時点の自動化状況（自動化推進Gのサポート対象）

工程 / プロジェクト	トレーサビリティ管理	ビルド	静的解析	単体テスト	結合テスト
プロジェクトA		■ →	■ →	■	
プロジェクトC	■ →	■ →	■ →	■ →	■
プロジェクトD			■ →	■ →	■
プロジェクトE		■ →	■ →	■	
プロジェクトF		■ →	■ →	■	
プロジェクトG	■ →	■ →	■		
プロジェクトH	■ →	■ →	■		

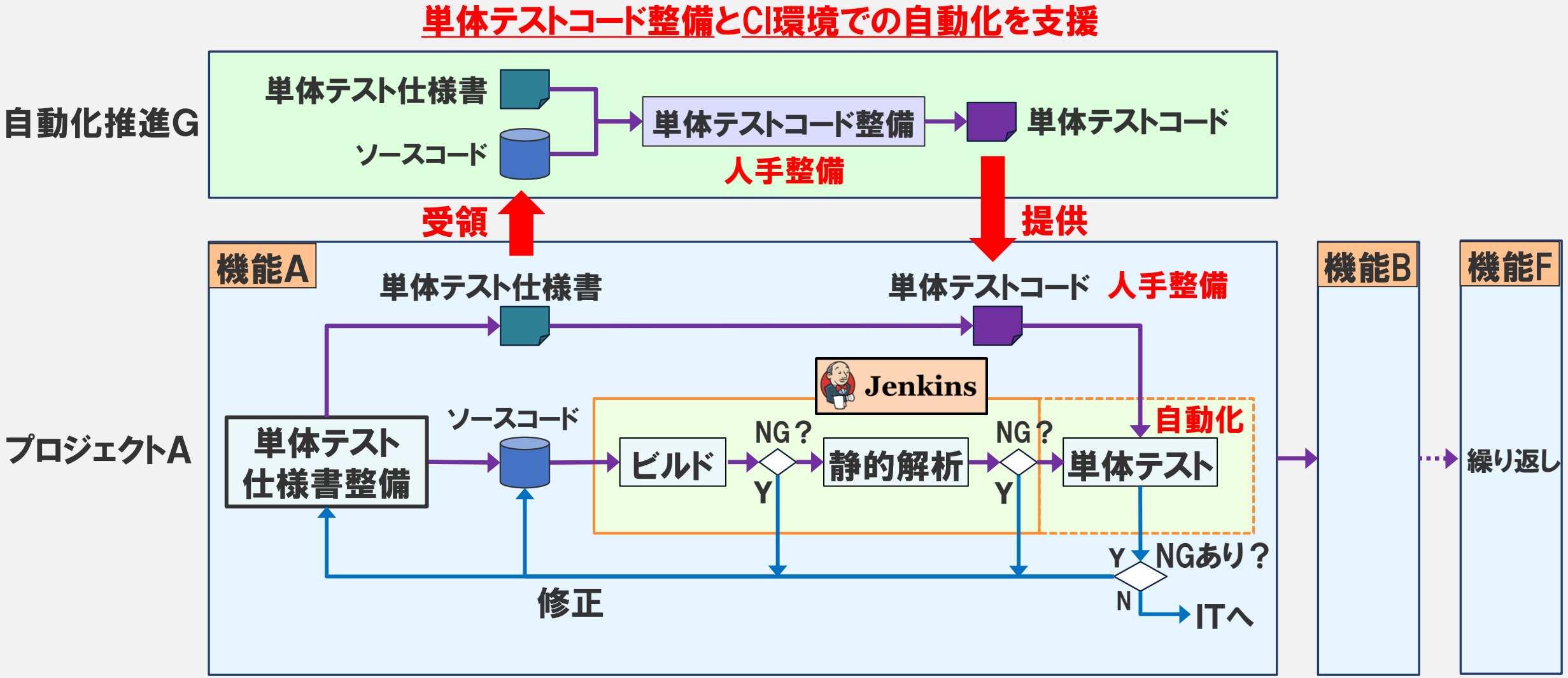
単体テスト自動化支援

※各工程のツールは
プロジェクトにより
異なる

単体テスト自動化支援

MORE ADVANCED

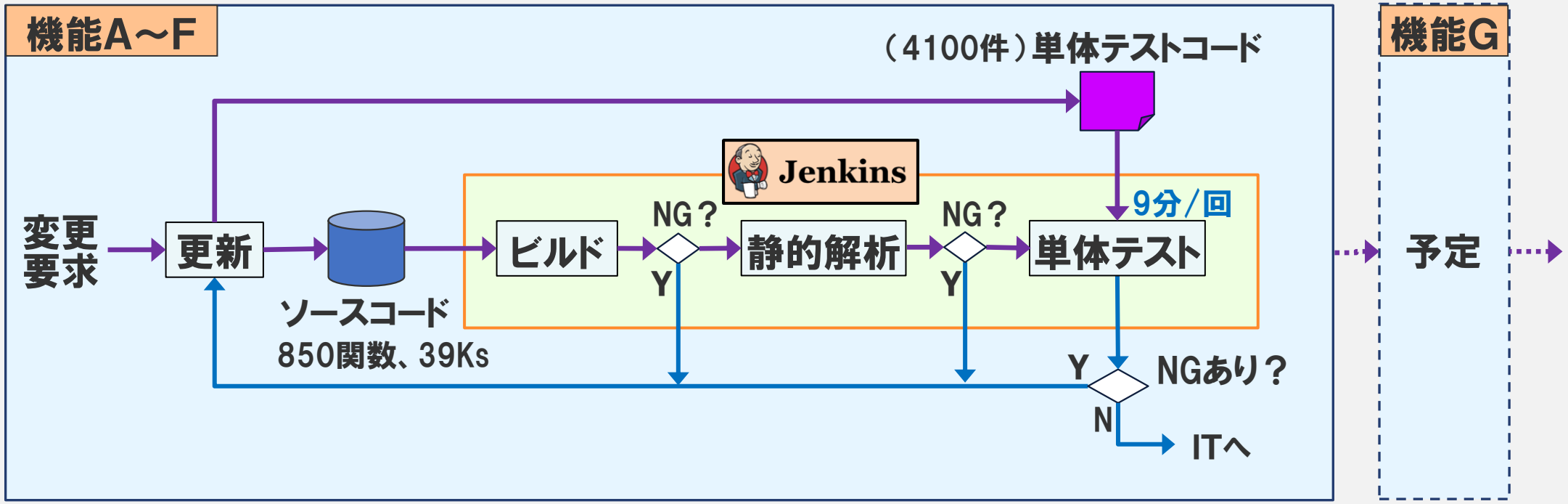
単体テスト自動化前　※インクリメンタル開発



気づいた課題

MORE ADVANCED

単体テスト自動化後



PL

単体テストコード整備に延べ5人月要しましたが、
ソースコード更新毎に単体テストが全件実施でき、
インクリメンタル開発が大幅に効率化できました

単体テストコード整備を
効率化できないか

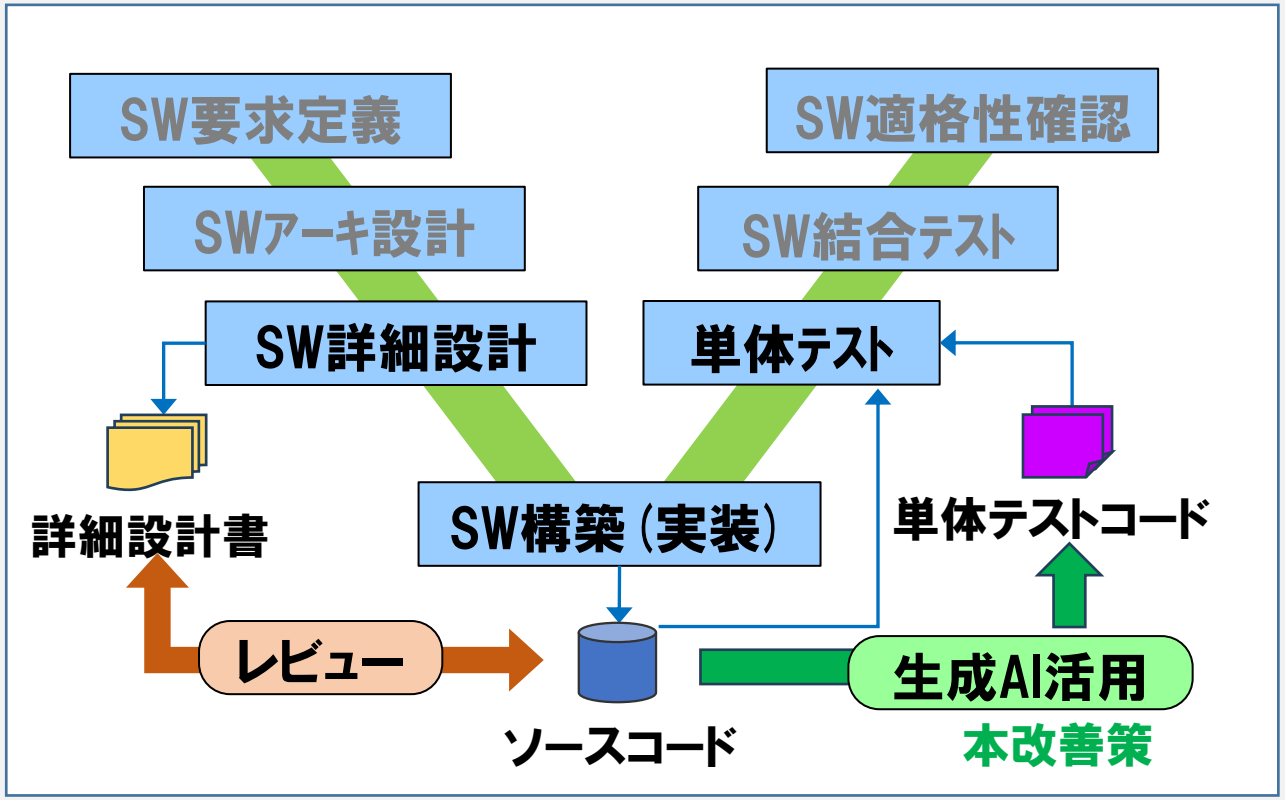
2．生成AI活用の試行錯誤

生成AIの活用検討

MORE ADVANCED

2023年度よりパナソニックグループでChatGPTベースのPX-AIが利用可能になっている。
生成AIを単体テストコードの整備に適用できないかと考えた。

[Panasonic Newsroom Japan](#) より（2023/8/7）



生成AI活用の試行錯誤

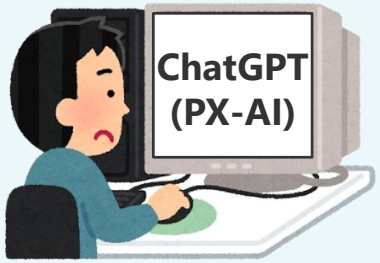
MORE ADVANCED

現場より単体テストコード整備を効率化したいとの相談を受け、生成AI活用を開始
(プロジェクトB)

1ケ月目:ChatGPT(PX-AI)

文字数制限があり、手修正が多いな..

C1 100%のテストコード生成は難しいな..



変更

2ケ月目:GitHub Copilot

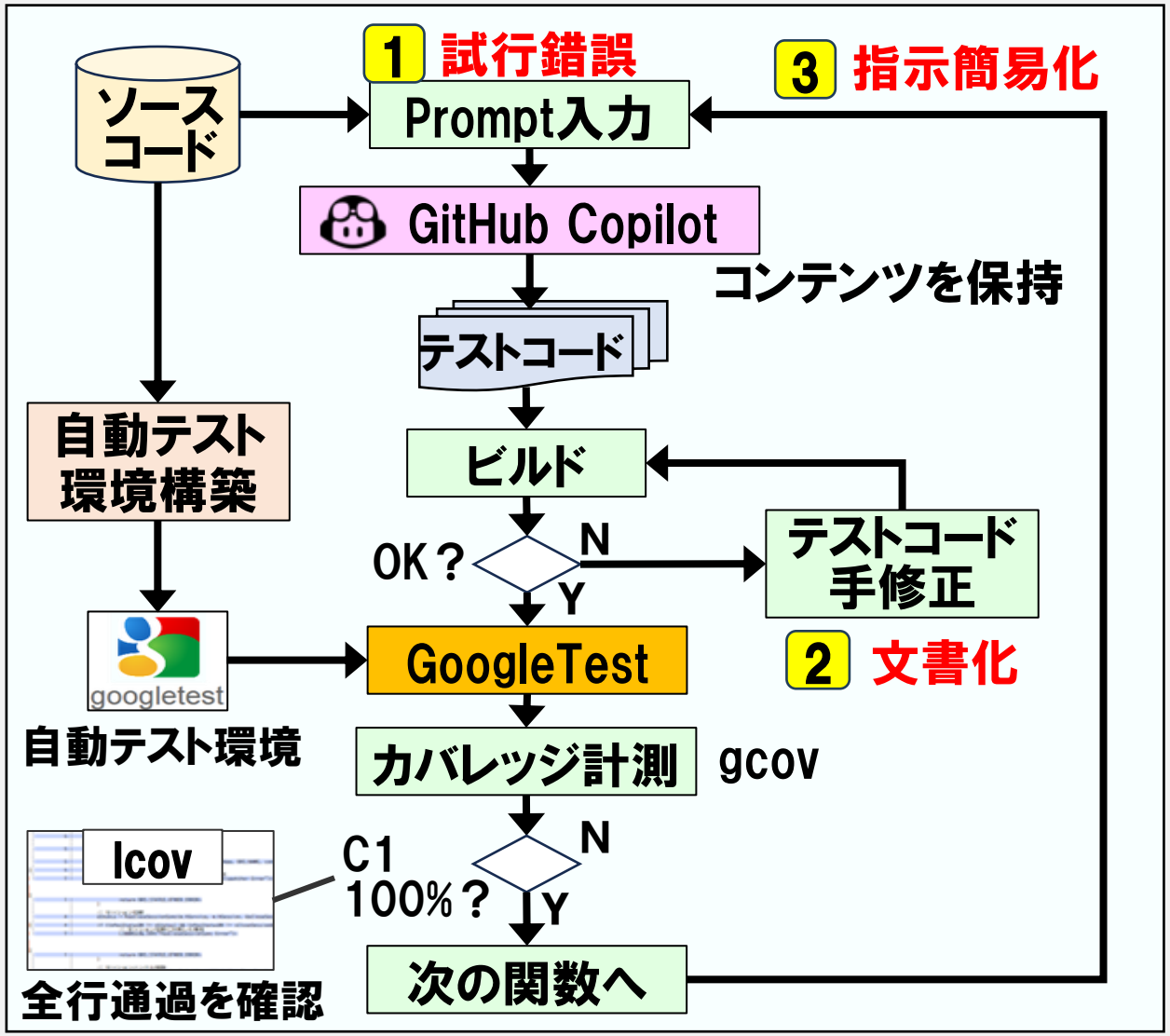
VSCode上で完結し、操作性が良い

手修正は必要だがC1 100%のテストコードが生成できる！

項目	ChatGPT (PX-AI)	GitHub Copilot
目的	特定の業務プロセスやデータ分析、意思決定支援など、**広範囲なビジネス用途に利用**	主に**プログラミング支援が目的**
機能	データ分析、予測、業務プロセスの自動化など、様々なビジネスニーズに応える	開発者にリアルタイムでコードの自動補完、関数提案などを行い、効率的なコーディングをサポート

最初からテストコード生成の用途に適した生成AIを選択すべきであった

ノウハウの蓄積と提供



1 試行錯誤しながら丁寧に指示

1) ○○関数に対してC1カバレッジの条件を、
同値分割、境界値分析に配慮して抽出して下さい
2) テストケースの具体例を示してください
3) C1カバレッジ100%を達成するためのGoogle Testと
Google Mockのコードを作成して下さい
4) テストコードにコメントを日本語で追加して下さい
5) MOCK_METHODをMOCK_METHODn（数字）の形式に
変更して下さい

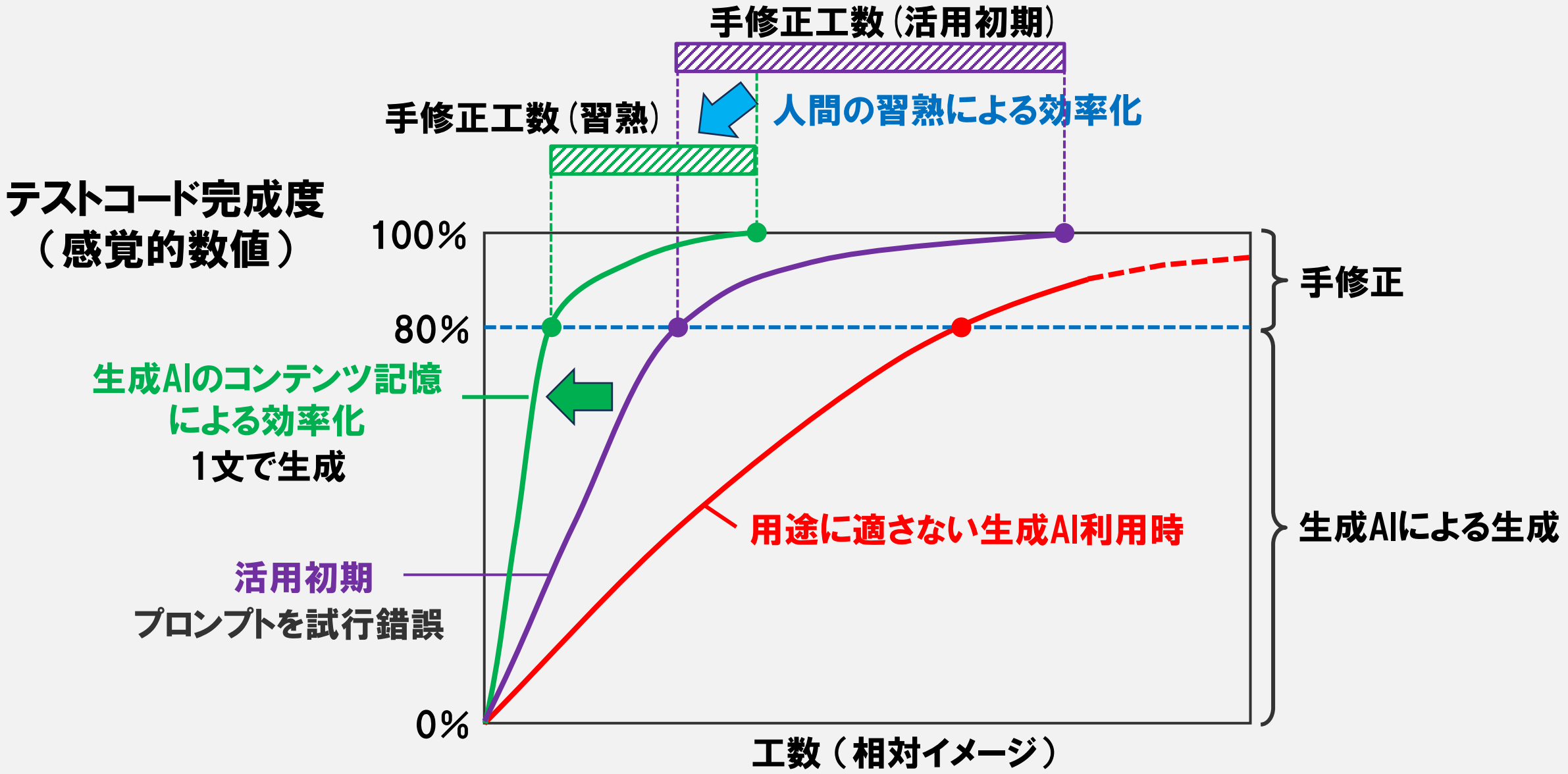
2 テストコードの手修正内容を文書化

・クラス内メソッドのMock化（仮想オブジェクト化）
・プライベートメンバ変数のテスト対応
・インライン展開の抑止　　など11項目

3 コンテンツは保持され、1文で生成可能に

・○○関数のC1カバレッジ100%を達成する
Google TestとGoogle Mockのコードを作成して下さい

生成AI活用によるテストコード完成度と工数



3．テストコード生成の効率化

テストコード生成の効率化-1

MORE ADVANCED

2つの施策を取り入れることとした

1. インストラクションファイルの設定

方針、要件、ルール等の
前提条件を記述

.github/copilot-instructions.md

100行強を記述

```
# コーディング規約
- コメントは必要に応じて日本語で記述すること。
# テスト基本方針
- Google Testを使用して、コードのテストを行うこと。
    :
# テストコード品質要件
- テストコードは、実装コードと同様に可読性を重視すること。
- テストケースは、C1(命令カバレッジ)100%を満たすこと。
    :
# テストケース記述ルール
- TEST_Fの直後にテスト概要、入力引数の値、リターン値を
  以下の例に倣って記述すること:
    :
# テスト対象関数の処理
# モックの実装
# モック対象関数の準備
# モック実装のstatic関数処理ルール
# 強いシンボルモック実装ルール
```

テストコード生成の効率化-2

MORE ADVANCED

2. Agentモードの活用

GitHub Copilot が自律的に動作し、指示に基づいて複数ステップの開発タスクを遂行

プロンプト

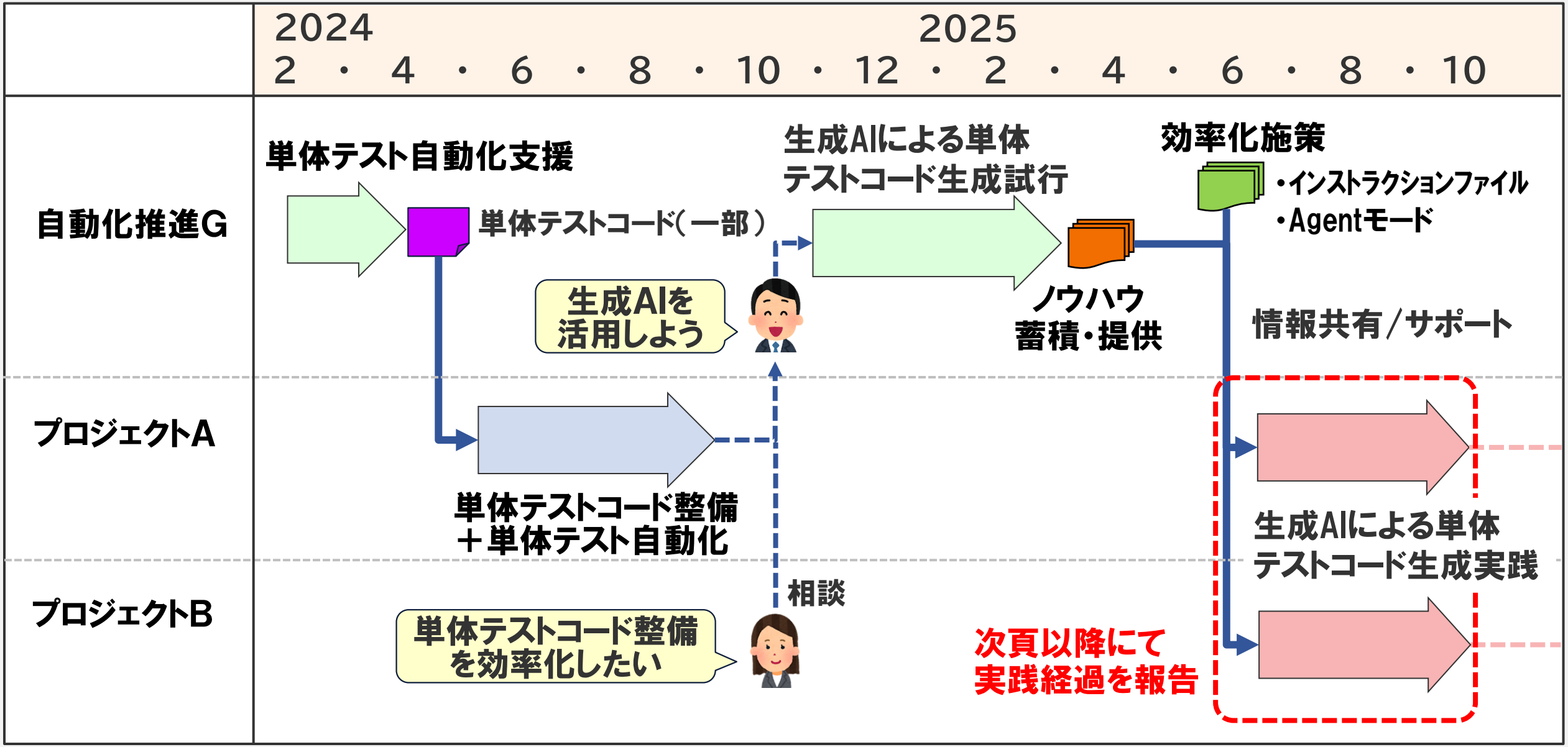
xxx関数のテストコードとモックコード
を生成してください。
また、Makefileの修正もお願いします

Agentモードのログ

xxx関数のテストコードとモックコードを生成いたします。
まず、現在のファイル状況を確認してから実装を進めます。
GoogleMock.h の 1 から 50 行を読み取ります。
：
xxx関数の依存関数のモック定義を追加します。
次に、Makefileに必要なstatic関数の弱シンボル化
設定を追加します。
：
ファイルの最後に関数のテストケースを追加します。
xxx関数のテストコードが正常に追加されました。
実装完了内容は以下です。
：

４．生成AIの実践活用と結果

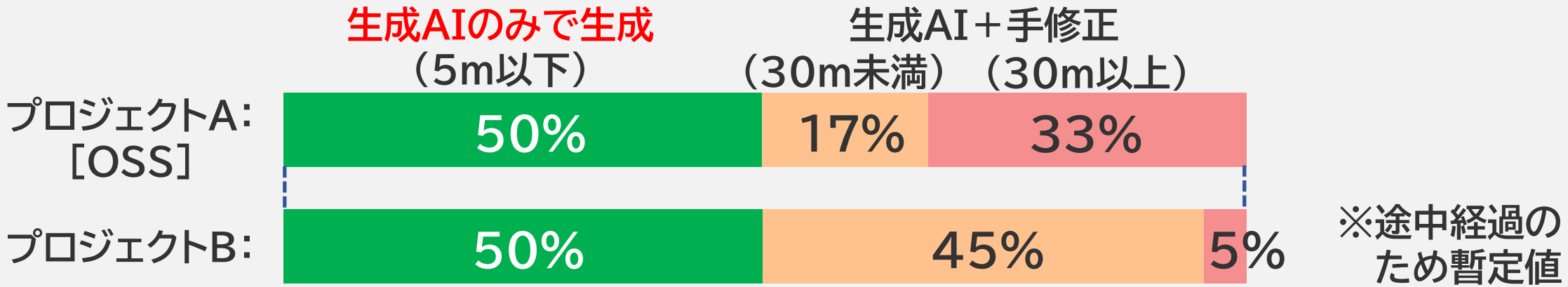
先行プロジェクトでの実践



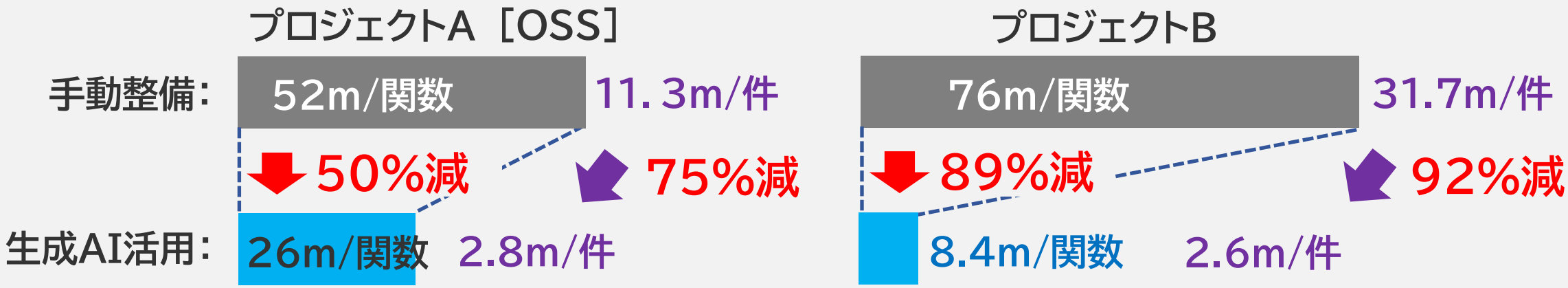
テストコード生成結果-1

MORE ADVANCED

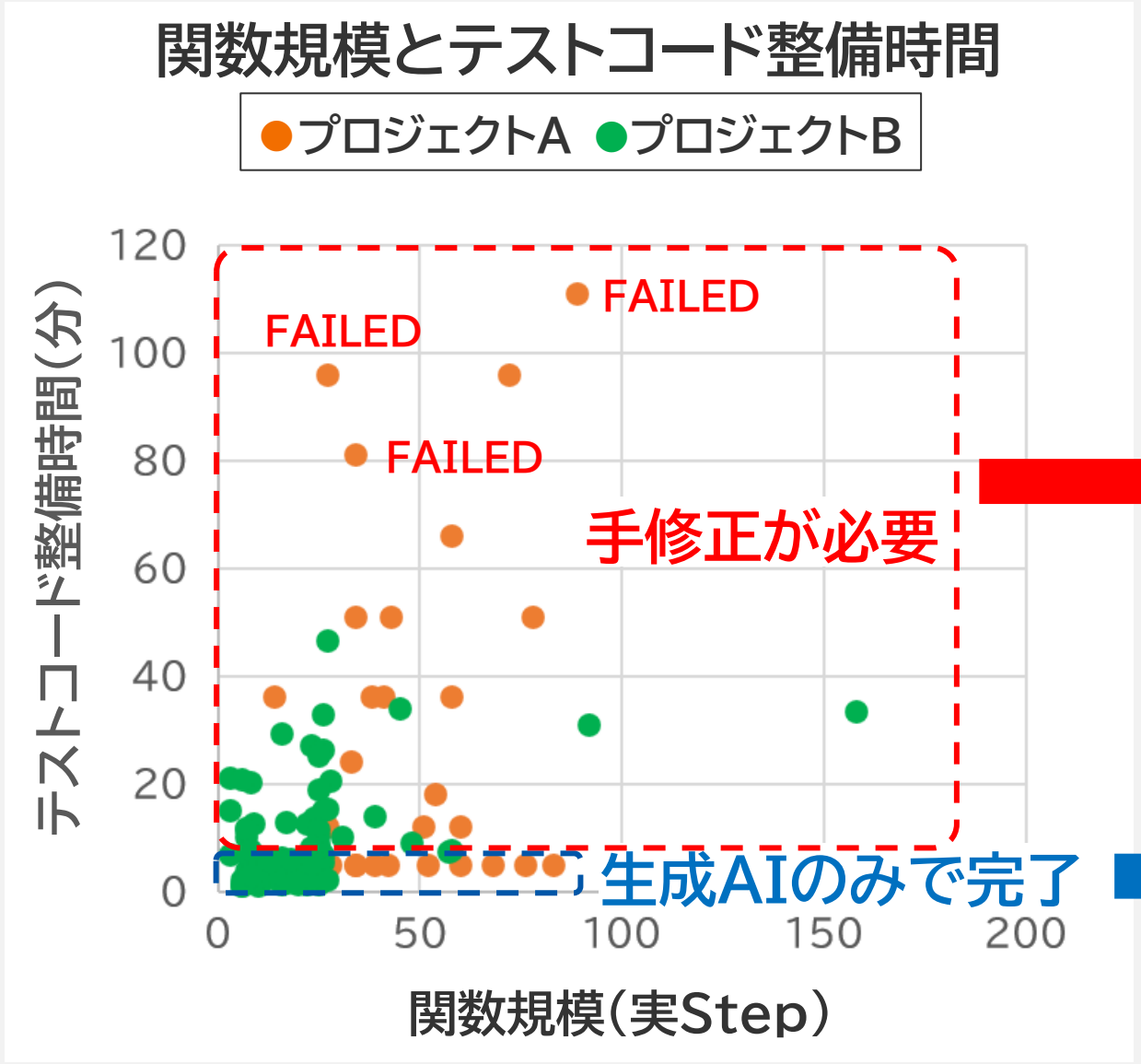
■生成AI活用による関数当たりのテストコード整備工数　　条件）C1カバレッジ100%



■テストコード生産性の比較　［関数（m/関数）、テストケース（m/件）］



テストコード生成結果-2



生成AIは関数規模が大きければ
時間がかかるという訳ではないが、
手修正時間は規模との相関がみられる

関数の特徴

- ポインタ渡し
- 複雑な構造体の処理
- FAILEDとなる関数
  - OSSでのテストコード/ソースコードの原因調査に時間を要した

- MOCKが不要な関数
- RETURNのみ返す関数

5．今後の予定

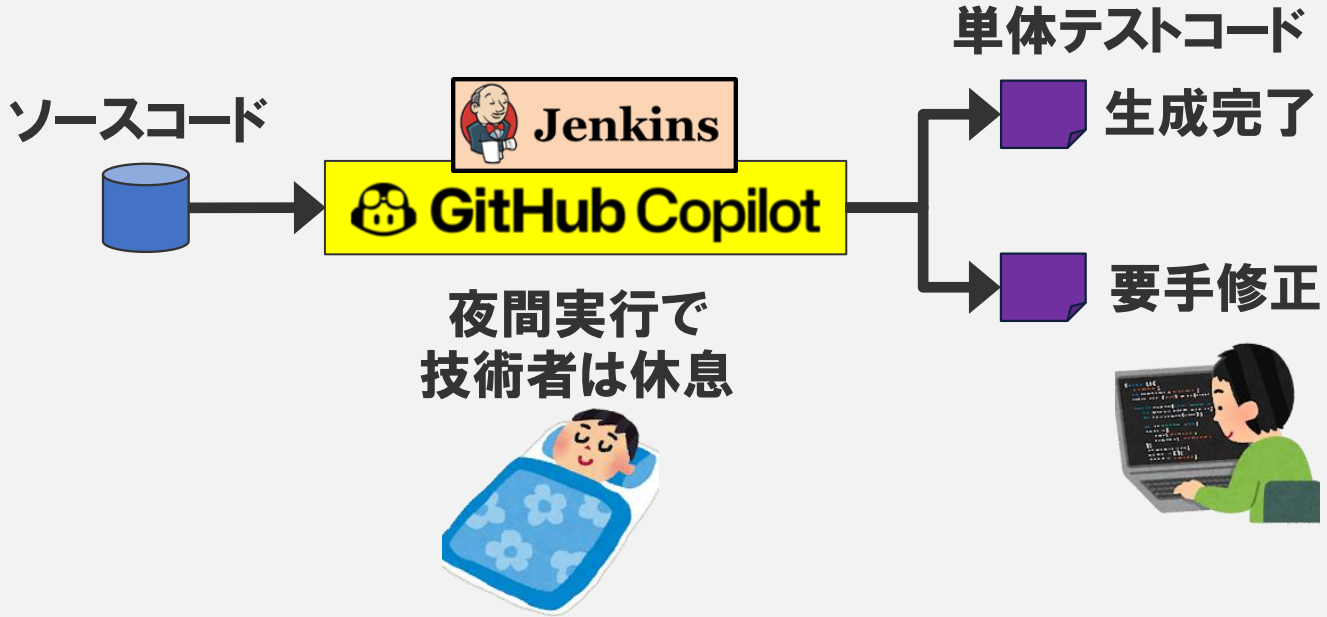
テストコード生成の更なる効率化に向けて

MORE ADVANCED

施策1 手修正内容を整理・ルール化し、インストラクションファイルを拡張
但し、生成AIでの完全性を追わず、手修正とのバランスを考慮

.github/copilot-instructions.md

施策2 CI環境にて自動生成させ、手修正が必要な関数を仕分けて対応

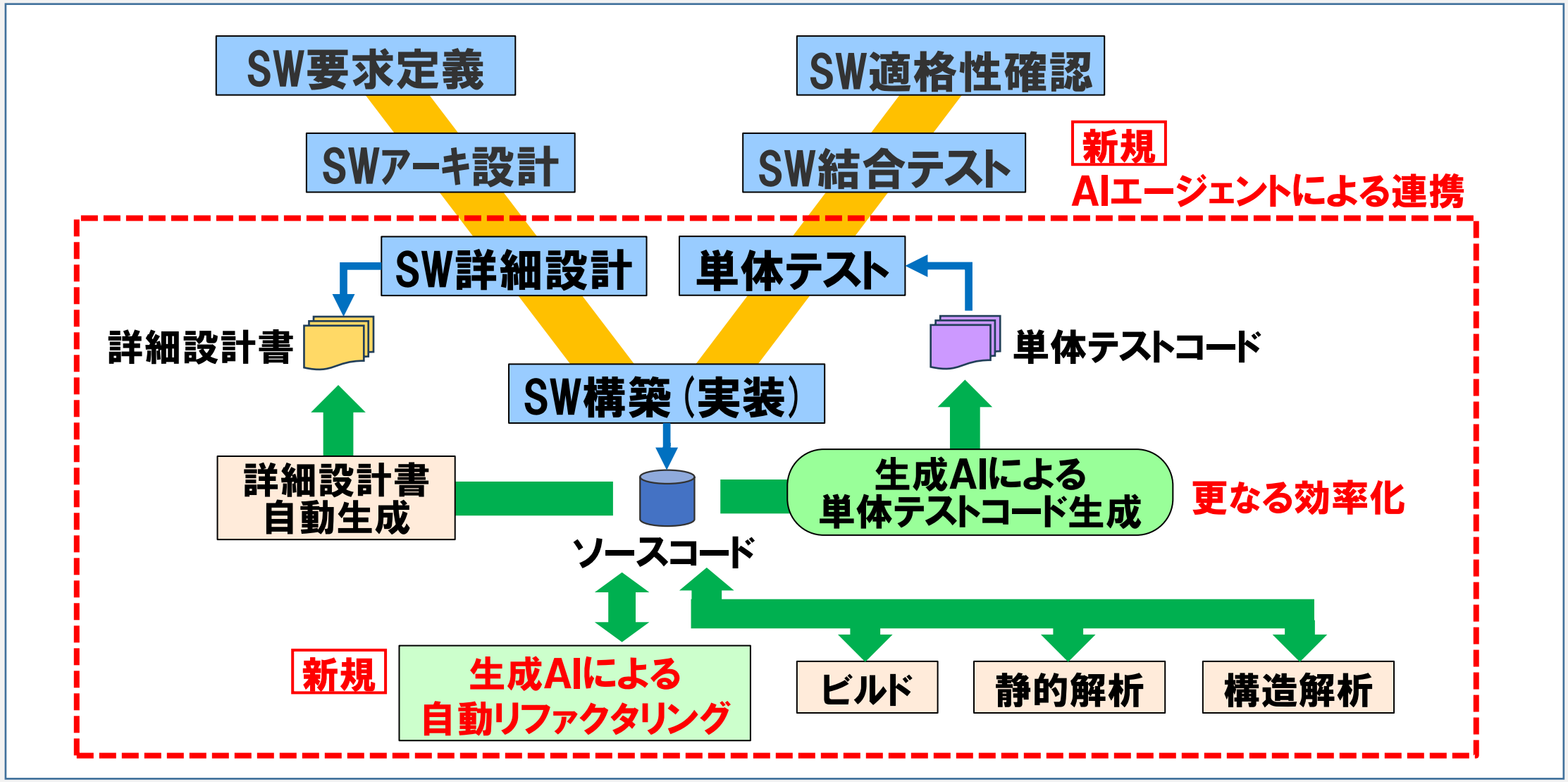


生成AI活用の展開

MORE ADVANCED

■生成AIの活用範囲の拡張と横展開を推進

現場を楽に



ご清聴ありがとうございました