

s3b_組み込みとwebソフトウェア開発 - 両現場から見たAIエージェント活用の現在地

WEBアプリ開発でAIと向き合った記録

2026/08/28 SWEST28
株式会社アルトワイズ 浜名将輝

自己紹介

氏名

浜名 将輝

略歴

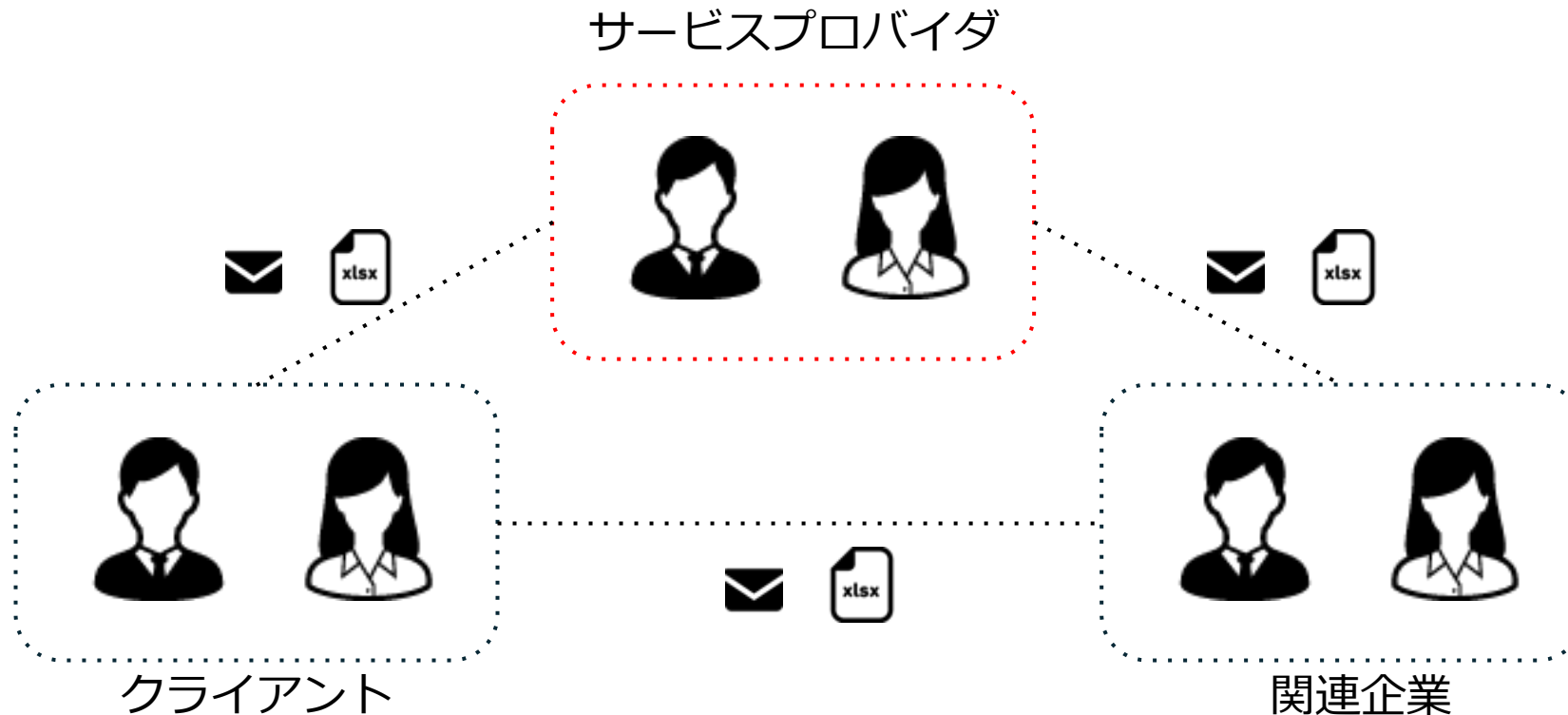
- 2020年 大学院修了
 - RISC-Vバイナリプログラムからの高位合成
 - LED-Campの実行委員
- 2020~2023年 新卒ではWEB業界へ
- 2023~2026年 コンサル会社へ転職
- 2026年4月~ 現職アルトワイズにて再びWEBエンジニアへ

やっていること

自社プロダクトを持つ企業の業務改善アプリの開発

課題

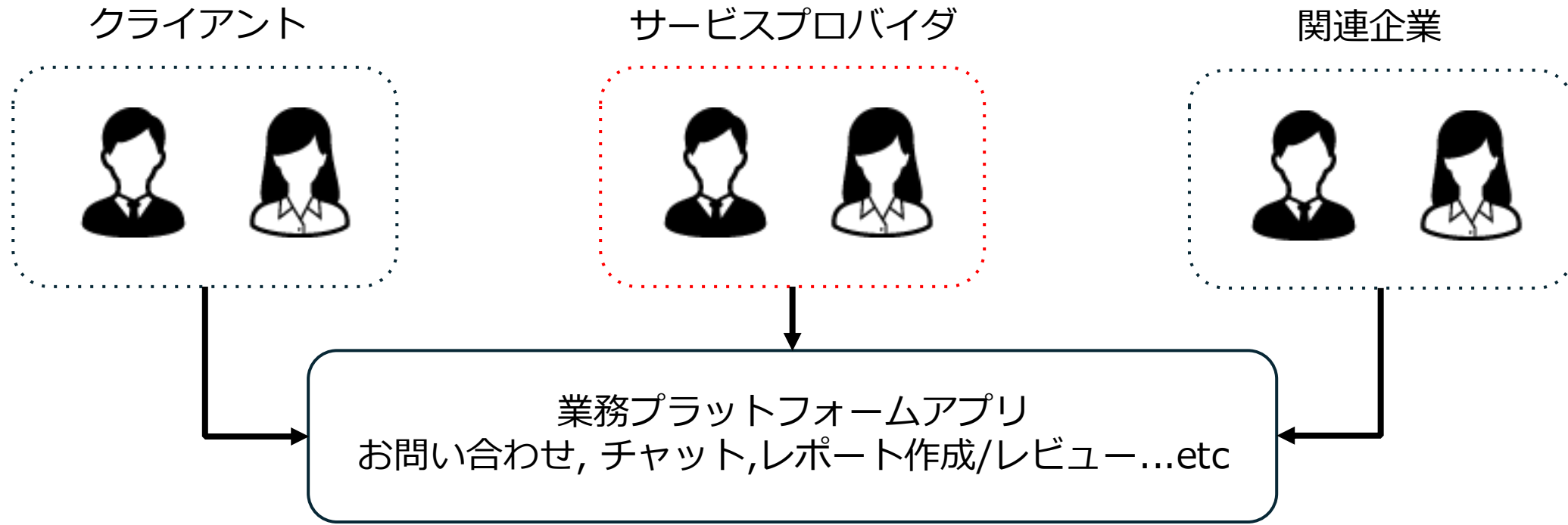
ステークホルダとのやり取りはメールやファイル共有がメイン
→業務中に得たナレッジが構造化されて蓄積できない



やっていること

自社プロダクトを持つ企業の業務改善アプリの開発

- コミュニケーション経路の一元化
- ナレッジの蓄積



開発ワークフローにAIを組み込んでいます

話すこと / 話さないこと

話すこと

- AIエージェントを開発に利用した際の気づき/感じたこと

話さないこと

- 特定のAIツールに絞った具体機能の深堀り・Tips
- チームや全社でのAIエージェント開発方法論, 導入方法論
- サービス自体にAI機能をどう組み込むか

開発初期(2026/04)

私の状態

- 前職(コンサル会社)で使っていたAIはTeamsのCopilot(対話)のみ
- プライベートではCursorを触ってみた程度

開発条件

- 材料はGenspark製モックアプリ
- 1ヶ月でフルスクラッチからのα版リリース
- 開発要員は自分1人

→AIが前提

リリース前後(2026/05)

私の状態

- VSCodeのClaude Code拡張で対話ベースで開発
- AI関連のYoutube動画を見始める
 - 世の中にはAIに役割分担をさせている人がいる
 - AIを社員のように扱っている人がいる

開発状況

- 初回リリースはなんとか間に合いそう
- リリース後にも増え続ける要望

ハーネスエンジニアリング(Open AIの事例;2026年2月)

Open AIの事例

- <https://openai.com/ja-JP/index/harness-engineeri>
- PRの起票・レビュー・マージまで全てエージェントが実施(PR1500件, 100万行)



「エージェントには、高度な目標に向けて進展を図るために必要なツール、抽象化、および内部構造が備わっていませんでした。私たちエンジニアリングチームの主な役割は、エージェントが有用な作業を行えるようにすることになりました。」

→ハーネスエンジニアリング

Agentic Codingにおける〇〇エンジニアリング

AIエージェントによる自律開発の風潮が強まっている

グラフエンジニアリング

2026年7月 単一ループから複数エージェント並列実行,分岐,チェックポイント等を組み込んだ自律ワークフローへの移行

ループエンジニアリング

2026年6月 AIエージェントにゴールを与え、検証を繰り返し、タスクを完遂させる実行系(ループ)の設計

ハーネスエンジニアリング

2026年初旬 AIエージェントにガイドライン, ツール, ガードレール, 権限等を与え、出力品質の下限を担保

コンテキストエンジニアリング

2025年 AIに与えるプロンプト群全体の最適化

プロンプトエンジニアリング

~2024年 AIに与えるプロンプト(1つ)の最適化

運用開始後(2026/06~07)

私の状態

- ハーネスやループの設計を開始

開発状況

- AIを自律で動かすほど、課題も見えてきた...
- 機能開発・改善は継続実施

Agentic Codingでの課題

AIエージェントの生成速度に人間の認知速度が追いつかない

- プログラムがどんどんブラックボックス化してくる
- そして不安になる

不安の中身

- バグがあるかもしれない
- セキュリティ的に問題があるかもしれない
- 保守性の低いコードになっているかもしれない



AIにも認知限界が存在する？

- LLMに明示的な指示を与えても常に守られるわけではない[Zhou 2023]
- 規約の数が増えるほど遵守率は下がる[Jiang 2024]
- 作業が長くなるほど規約の遵守率は下がる[Laban 2025]

→ルールが守られるのはあくまでも確率に従う

※モデルの性能が上がるにつれて遵守確率は上がる傾向にあるが保証にはならない

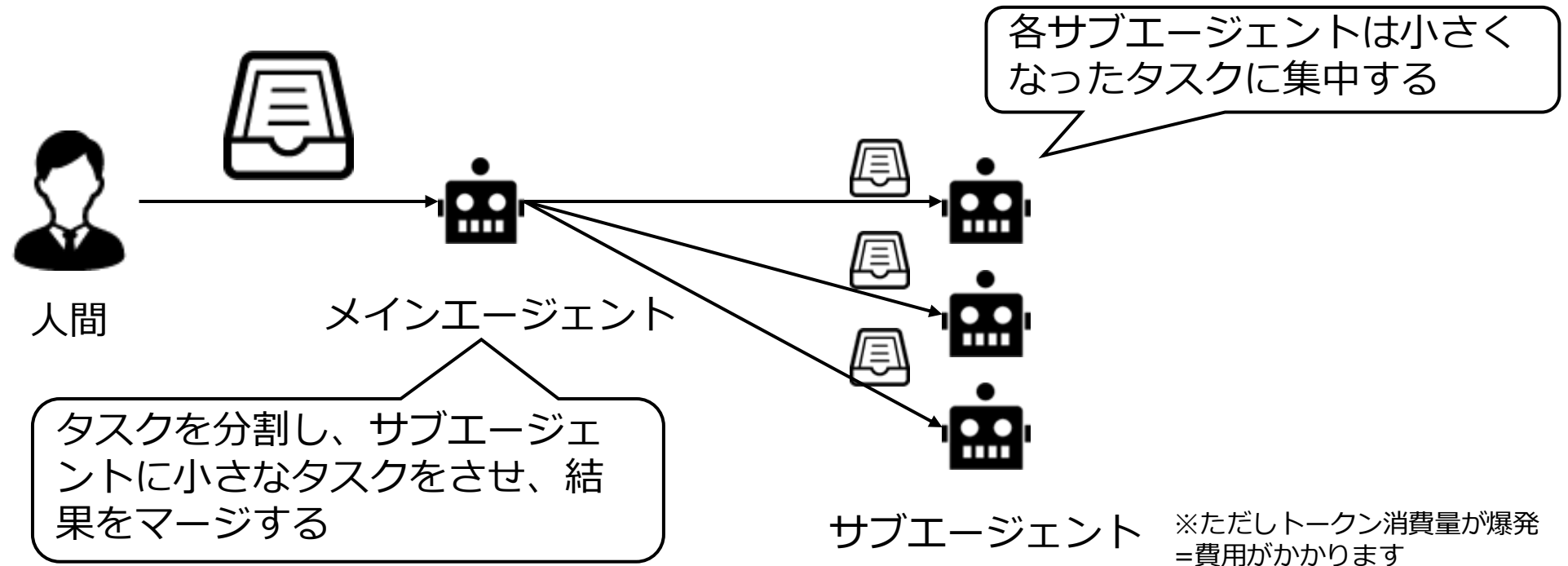
参考文献

- [Zhou 2023] <https://arxiv.org/abs/2311.07911>
- [Jiang 2024] <https://arxiv.org/abs/2310.20410>
- [Laban 2025] <https://arxiv.org/abs/2505.06120>

AIの専門化による出力品質の向上

AIが扱うコンテキストを絞り、作業量を減らすと品質・精度が上がる

- 間接事例: Claude Codeマルチエージェントシステム, Dynamic Workflow



参考文献

<https://claude.com/blog/building-multi-agent-systems-when-and-how-to-use-them>

AIにコードレビューさせてみた

不安の中身をレビュー観点に落として規約化してみた

- それでも不安は消えない
- コードレビューも確率的な出力であり、品質の決定的な保証にならない
- 結局ブラックボックス化していく

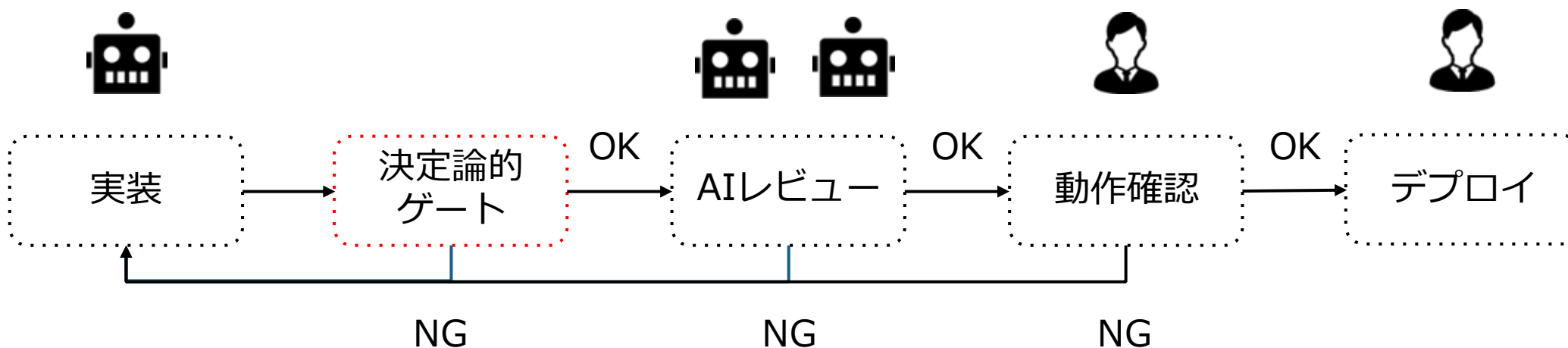
安心はどのようにして得られるのだろうか

- 決定的な品質保証の仕組みを作ること

決定論的ゲート

プロダクトコードそのものは量が多く認知負荷が高

- ・ 機械的に判断できるところは決定論的ゲート化
- ・ ゲートを通れば、ゲートに書いてあることは決定的に保証できる



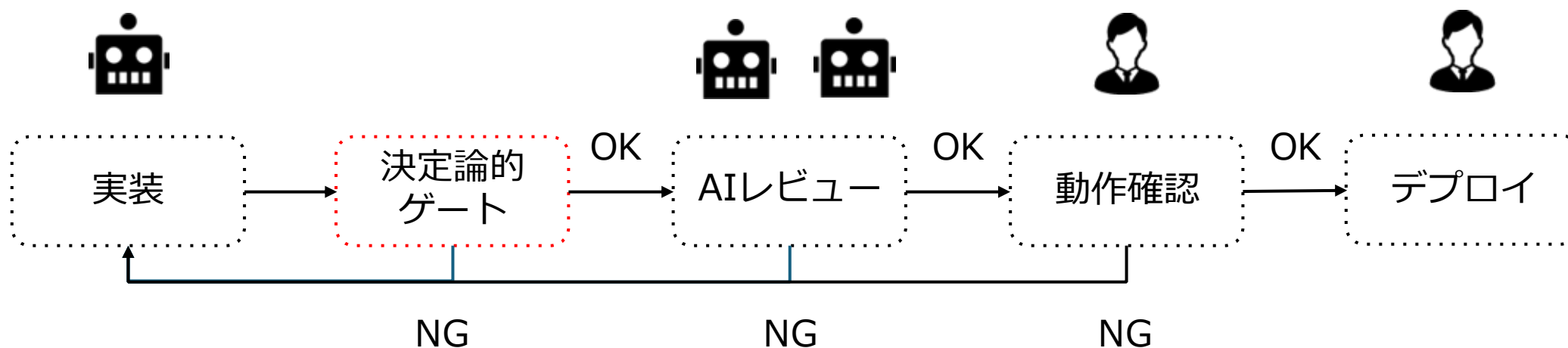
例)

- ・ 自動テスト(vitest)
- ・ ビルドのドライラン, 型チェック(tsc)
- ・ 依存ライブラリ脆弱性スキャン(npm audit)
- ・ 脆弱性パターン検出(SAST; Semgrep)
- ・ シークレット検出(gitleaks)
- ・ OSSライセンスチェック
- ・ ドキュメントフォーマットチェック(remark)
- ・ リンタ(ESLint)
- ・ コード内依存チェック(dependency-cruiser)
- ...など

決定論的ゲート

AIが読むべき規約・ルールも決定論的ゲートに移行できるものは移行する

- ・ コンテキスト圧迫を抑え出力品質向上、トークン削減に寄与



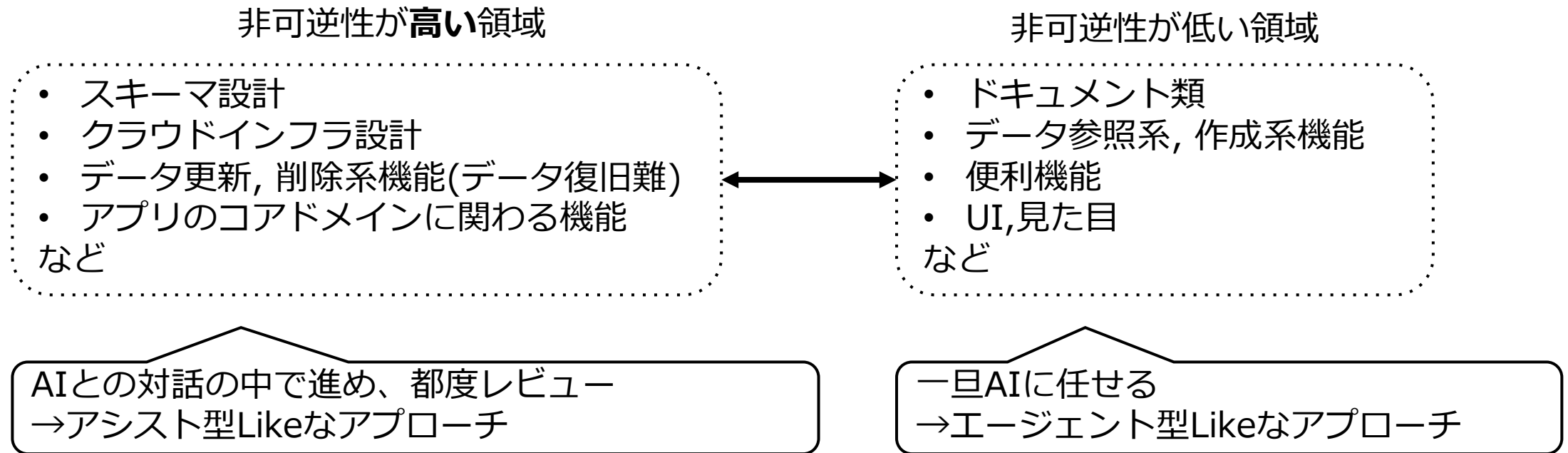
例)

- ・ 自動テスト(vitest)
- ・ ビルドのドライラン, 型チェック(tsc)
- ・ 依存ライブラリ脆弱性スキャン(npm audit)
- ・ 脆弱性パターン検出(SAST; Semgrep)
- ・ シークレット検出(gitleaks)
- ・ OSSライセンスチェック
- ・ ドキュメントフォーマットチェック(remark)
- ・ リンタ(ESLint)
- ・ コード内依存チェック(dependency-cruiser)
- ...など

戻せるか？戻せないか？(戻しづらいか？)

人間の認知リソースを非可逆性が高いところに持っていく

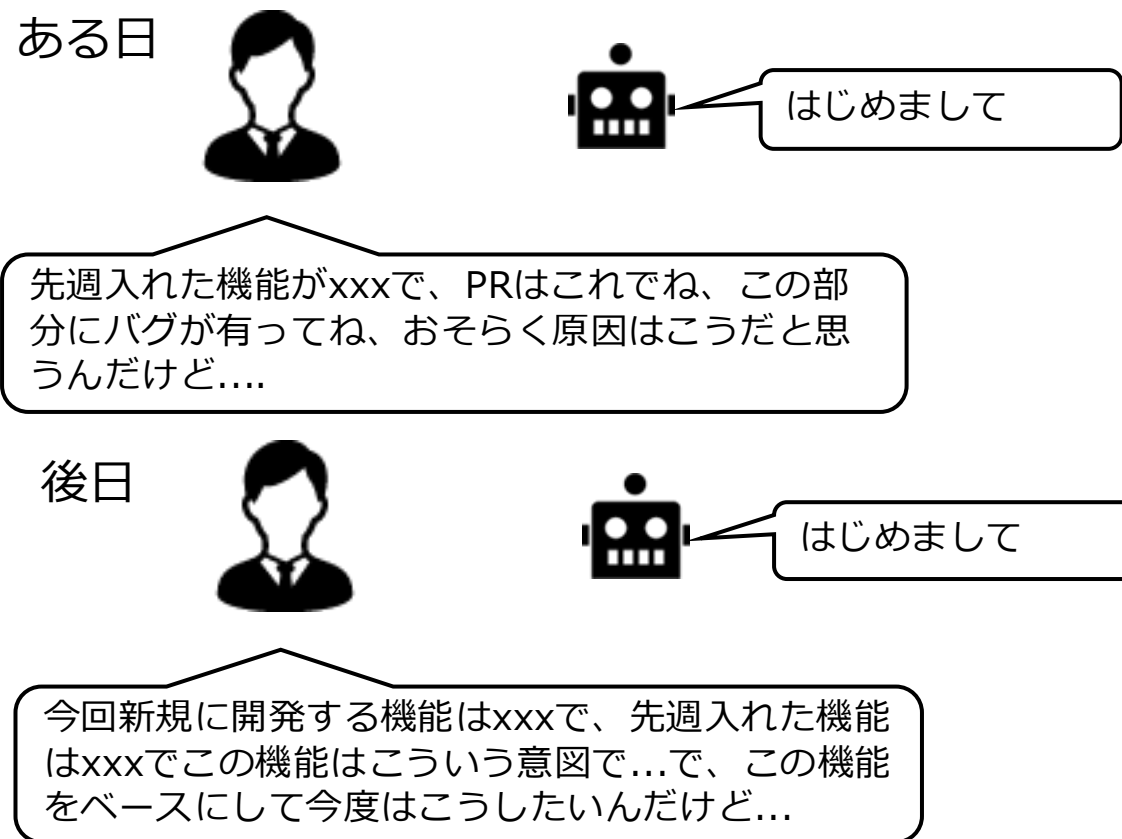
- コード生成コスト自体は激減したので戻せるか？戻せないか？が重要に



人間チームとAIチームの空気感の違い

チームとしての共通認識、空気感が育たない

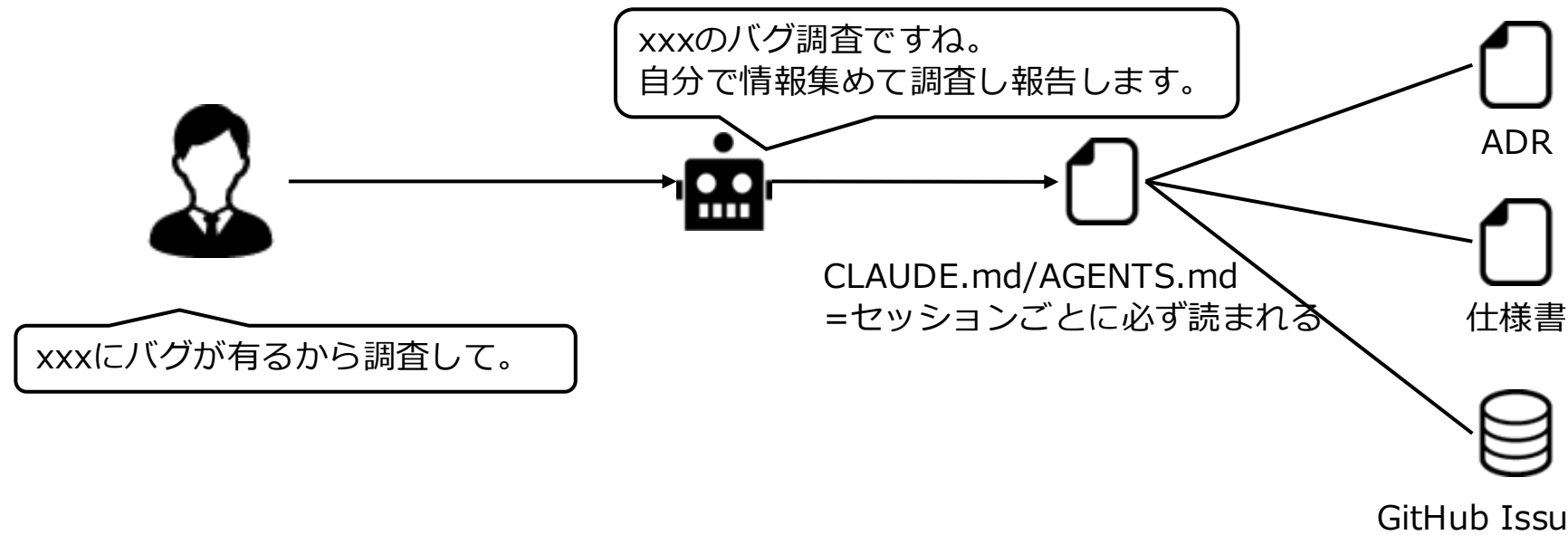
- ・ 人間は記憶を蓄積することができる
- ・ AIは毎回記憶をリセットされる



AIと歴史を刻む

過去の判断、経緯、作業履歴を構造化してすべて保存する

- ADR(Architecture Decision Record)
 - 設計論点をまとめ、各論点に対して採用, 不採用を理由付きで残す
- 文脈エンジン(GitHubなど)
 - 過去のタスク(Issue)や、レビュー履歴(PullRequest)、経緯を残す
- CLAUDE.md, AGENTS.mdは各種資料のインデックスに徹する



現在どう運用しているか

Claude Codeを使用

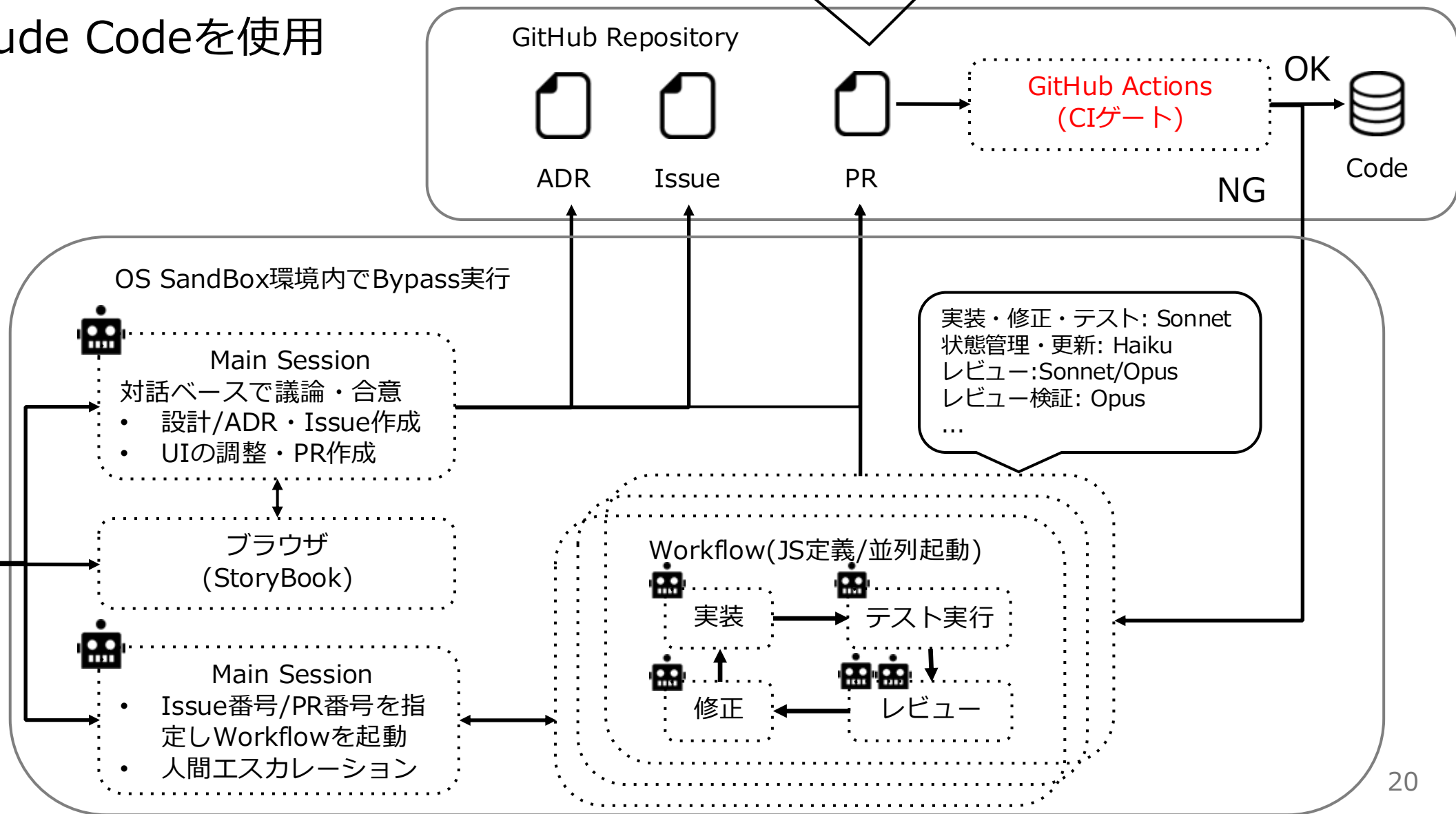
人間が見るPRはパスベースで判定・マーク

- スキーマ変更
- AWS CDK変更
- など...

営業・顧客

要望

私



AIによって変わったと感ずること

- エンジニアの役割が製品自体の開発から、品質保証の仕組みづくりへ
- 課題そのものを思考する時間が増えてきた
- 領域ごとの役割分担が粗くなってきた

AIエージェントの組み込み開発への適用 2026年夏

2026/08/28 SWEST28 s3b
島津トラステック 桐畑 鷹輔

自己紹介

桐畑 鷹輔（きりはた ようすけ）

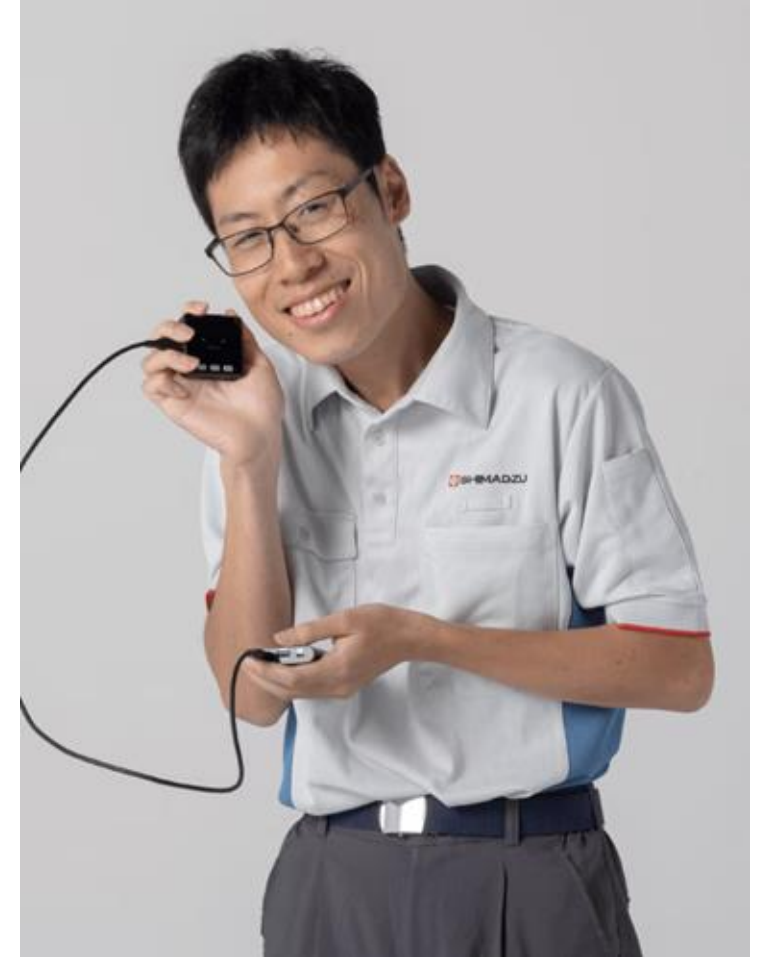
分析計測装置・産機・航空機器の組込み開発を経験
現在は、研究開発部門（新事業、バイオ、etc...）
を中心に幅広くソフトに関係する技術支援を実施

魔改造の夜 ビニール傘滞空マッチ 制御班

趣味：電子工作

好きなAIエージェント：Claude Code

本発表は、一個人によるものであり、所属組織は関係ありません



生成AIの変遷

段階	主な利用形体	人の関わり方
プロンプト（2020～）	チャット型	毎回入力し、結果を閲覧（コピペ）
コンテキスト（2025～）	エディタ統合 （VSCode、Cursor）	各リクエストの応答、差分を見て承認や修正依頼
ハーネス（2026初～）	CLI（Claude Code、Codex）	タスク単位で作業を依頼
ループ（2026中～）	CLI、CI、バックグラウンド	停止条件と検証結果を閲覧

2026年春の試行：携帯ゲーム機のLinuxメインライン化

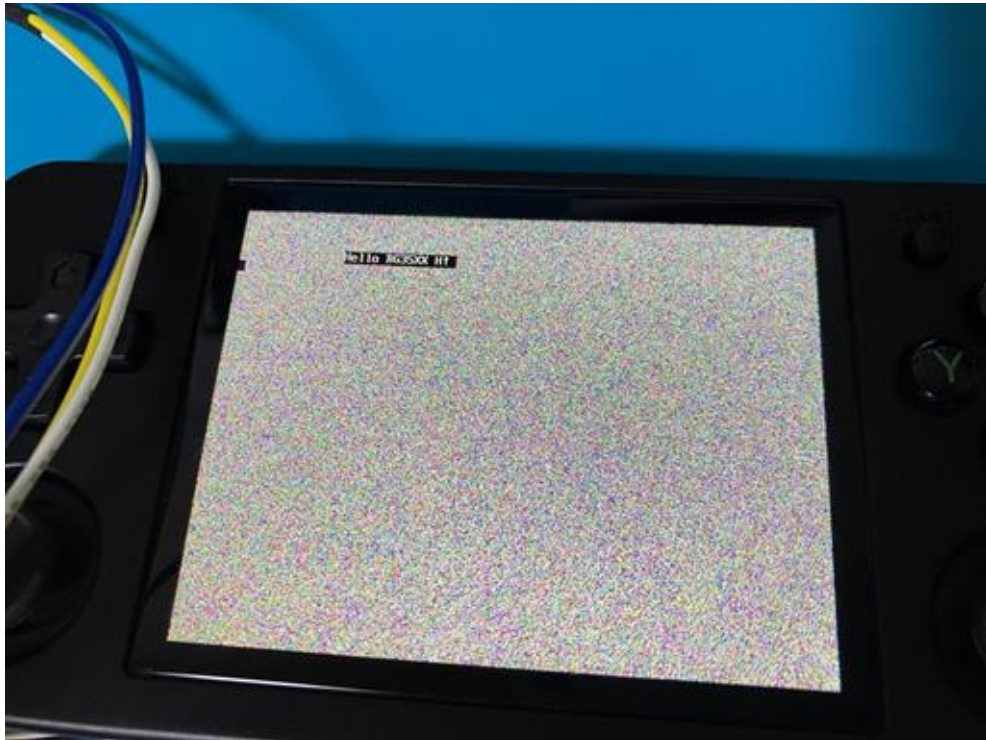
携帯ゲーム機 RG35XXH でメインラインLinuxを動作させることを題材に試行

AIエージェントに任せたこと

- ・ 環境構築、技術調査
- ・ ブートローダ、Linuxカーネル、Ubuntuユーザーランドのビルド
- ・ 不足ドライバの有効化・修正、デバイスツリーの記述
- ・ ログ解析

人間（自身）が行ったこと

- ・ SDカードの差替え、電源ON/OFF、コマンド実行・ログのフィードバック



組込み開発でAIを自律的に利用するには

- ・組込みは、ソフトウェアで完結せず、ハードウェアを含めた検証が必要

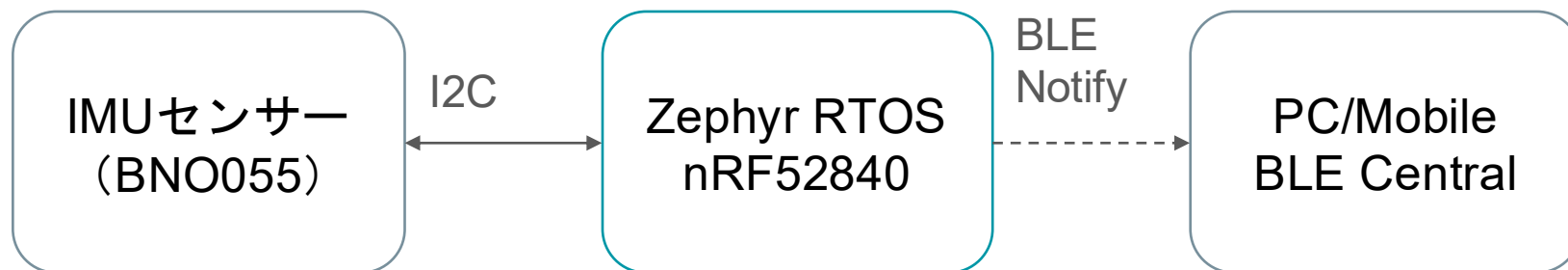
→もっとAIに任せられないか？

→AIに直接ハードウェアを触らせることはもうできそう（～CLI、Skills、etc）

→より組込み開発らしい題材で試行を決意

題材紹介：IMU姿勢情報をBLE送信するファームウェア

- ・ センサーデータの取得
- ・ BLE通信の送受信

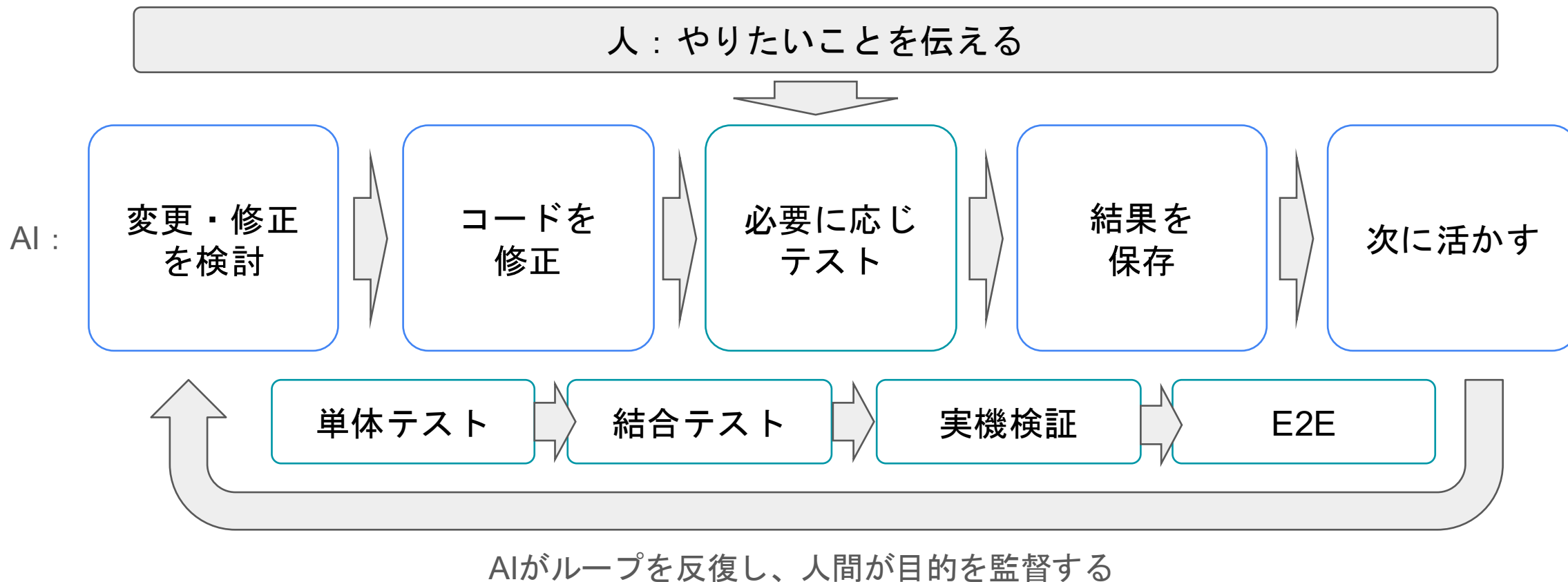


選定・利用したハードウェアとソフトウェア

- ・ ハードウェア（実行環境）：nRF52840 Adafruit Feather・BNO055 IMU
- ・ デバッガ：J-Link（nRF52840 DK）
- ・ ログ出力：SEGGER RTT
- ・ RTOS：Zephyr
- ・ 単体テスト：ztest（Zephyr公式サポート テストフレームワーク）
- ・ 結合テスト：Renode（Zephyr公式サポート エミュレータ）
- ・ E2Eテスト：Python、bleak（Python BLEライブラリ）
- ・ AIエージェント：Codex（Claude Codeでの動作も可能）

今回作成したループのイメージ

AGENTS.mdとSkillsで以下を実現



4つの検証層



4. E2E : BLE Centralと実際に通信して観測

3. 実機検証 : 書込み・ログを観測

2. 結合テスト : 模擬ペリフェラル (I2C, etc...) ・ センサ模擬

1. 単体テスト : 純粋なアプリロジック

層が上がるほど、実世界に近づく＝時間・環境依存が増える
検証層を増やすのは、失敗を早く検出するため

各検証層で証明できること・できないこと

検証層	証明できること	証明できないこと
1. 単体テスト	純粋なロジック、単体テスト	ペリフェラル、実時間、消費電力
2. 結合テスト	nRF52840モデル上でのZephyr起動、BLE初期化、Advertising開始処理、模擬ペリフェラル処理	実センサーの応答、実機タイミング
3. 実機検証	ビルド、J-Link経由の書込、実機起動、BLE初期化、RTT起動ログ	Centralが必要なBLE通信の実際 (Advertising、GATT、ペイロード)
4. E2E	Centralが観測するAdvertising、GATT、ペイロード	E2Eシナリオ外の挙動 (消費電力、無線性能、姿勢精度)

今回実施した検証層とループの構築手順

- ① 最低限の合格条件を決定：何が確認できたら「正しく動いた」とするかを決める
- ② 繰り返し可能な検証方法を固定：対象環境、ツール、実行方法、結果の確認方法をそろえる
- ③ ロジックの検証を作成：変更 → テスト → 結果確認 → 修正を繰り返す

——最小限の検証層を含むループが完成——

- ④ 必要な検証層を追加：シミュレータ → 実機 → 外部からの確認へ検証範囲を広げる
- ⑤ 結果から改善：見つかった問題を、可能な限り安い検証層のテストへ追加する

——AIがループを反復し、人間が目的・合格条件・安全上の判断を監督——

検証層の構築手順① 最低限の合格条件を決定

- ・ 目的：「正しく動いた」を機械判定可能にする
- ・ 人の役割：ファームウェアとして最低限の動作仕様を伝達
- ・ AIへの依頼：”起動からAdvertising開始までの合格条件を定義せよ”
- ・ AIの成果物：成功条件の定義、ログ出力処理、自動判定設定
- ・ 検証結果（判定実行後）：成功マーカの有無による合否
- ・ 可能になること：AIが毎回同じ基準で動作を判定できる

検証層の構築手順② 再現可能な検証方法を固定

- ・ 目的：誰がいつ実行しても同じ条件で検証できるようにする
- ・ 人の役割：ツール、利用条件を決める
- ・ AIへの依頼：“依存関係、操作、結果形式を固定せよ、AGENT.md、Skills”
 - ・ AIの作業：ビルド・書込み・ログ取得を共通操作にまとめる
- ・ AIの成果物：環境設定、依存関係、操作自動化Makefile・スクリプト
- ・ 検証結果（環境確認・ビルド後）：バージョン確認結果とビルド結果
- ・ 可能になること：AIが環境やコマンドを推測せず検証を再現できる

検証層の構築手順③ ロジックの検証を作成

- ・ 目的：実機なしで回る最小の検証ループを作る
- ・ 人の役割：期待する入出力と異常時の動作を決める
- ・ AIへの依頼：“ロジックを分離し、正常・境界・異常テストを作成せよ”
 - ・ AIの作業：センサー変換やBLEデータ生成を分離する
- ・ AIの成果物：データ変換処理、BLEデータ生成処理、単体テスト
- ・ 検証結果（単体テスト後）：テストの可否と実行された処理の範囲
- ・ 可能になること：AIが実機なしでロジック不具合を修正できる

検証層の構築手順④ 必要な検証層を追加

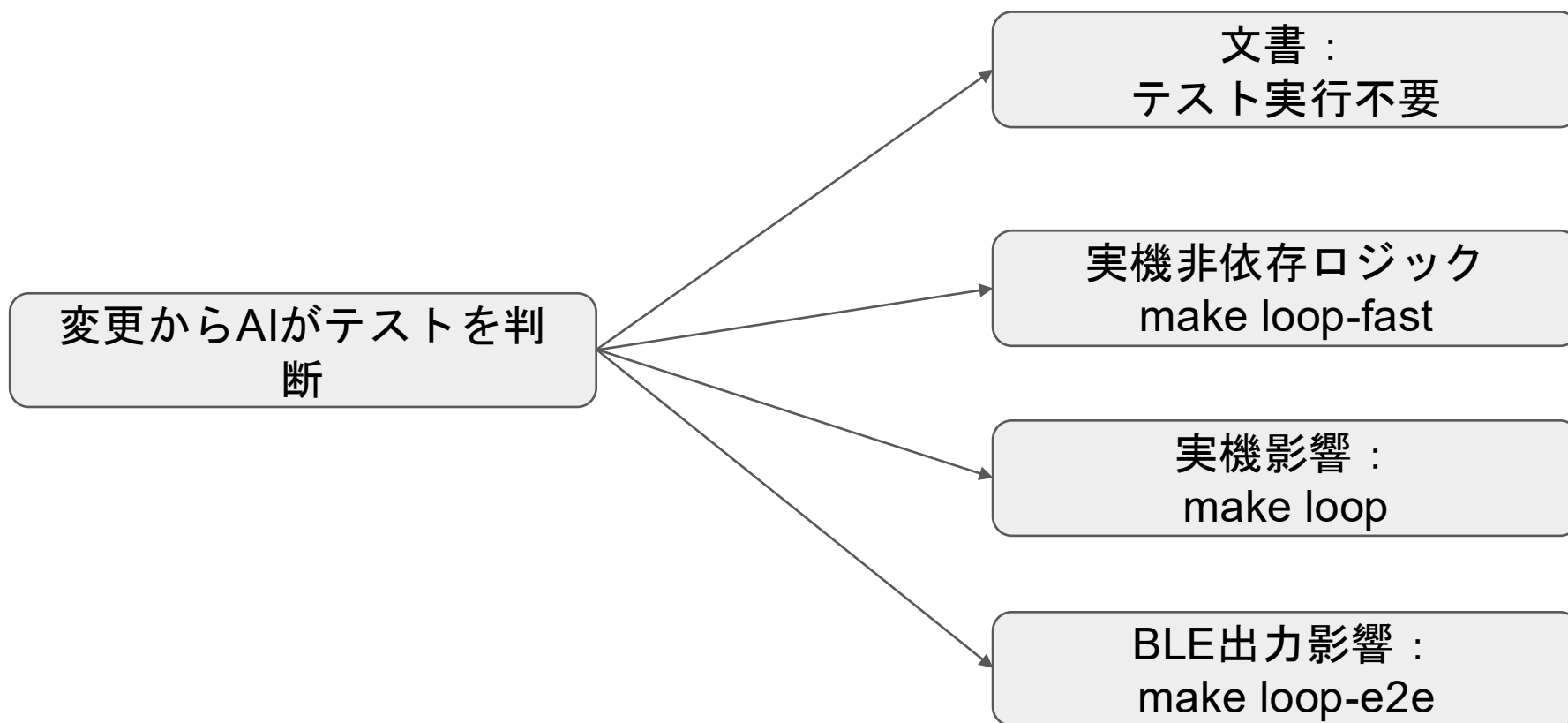
- ・ 目的：単体テストだけでは確認できない動作を上位層で補う
- ・ 人の役割：実機の準備、配線、電源などの物理操作を行う
- ・ AIへの依頼：結合テスト/実機検証/実BLE通信E2Eを追加せよ
 - ・ AIの作業①：仮想センサーとシミュレータテストを作成する
 - ・ AIの作業②：実機起動とホストからのBLE確認を自動化する
- ・ AIの成果物：IMUドライバ、BLE通信処理、各層のテスト
- ・ 検証結果（各層の実行後）：シミュレータ・実機・BLEそれぞれの合否
- ・ 可能になること：問題が発生した検証層を切り分けられる

検証層の構築手順⑤ 結果から改善

- ・ 目的：実機で得た知見（失敗対策）を、短時間で実行できるテストへ戻す
- ・ 人の役割：実機状態、許容値、安全性について判断する
- ・ AIへの依頼：実機で見つけた問題を下位層で再現可能にせよ
 - ・ AIの作業①：ログから原因、確認方法、再現条件を特定する
 - ・ AIの作業②：故障の再現機能と再発防止テストを追加する
- ・ AIの成果物：仮想センサー、故障注入機能、再発防止テスト
- ・ 検証結果（追加テスト実行後）：問題の再現と修正後の合格結果
- ・ 可能になること：一度見つけた問題を今後、繰り返さない

ループを回す契機

AGENT.md、Skillsが編集影響範囲から実行する内容を判断・ツールを呼出



デモ

- ・ 作成したファームウェアとスマートフォンアプリの実行の様子をご覧ください
- ・ プロジェクト内のAGENTS.mdとSkillsを簡単にご覧ください
- ・ 実際にAIエージェントへの依頼～ループ実行の様子をご覧ください

“loop-engineering BNO055由来の異常な姿勢値がBLEへ通知されないようにせよ”

まとめ

- ・ AIエージェントは、すでにハードウェアを含めた自律ループを構築可能

＜AIエージェントを組み込み開発へ活用するための要点：自身の現在地2026年夏＞

AIエージェントが操作・観測・証跡を扱いやすい形に開発環境を構築すること

- と
 - ・ 操作：ツールの呼出
 - ・ 観測：ログ出力
 - ・ 証跡：設計判断根拠ADR

→AIエージェントが、これそのものを行ってくれる