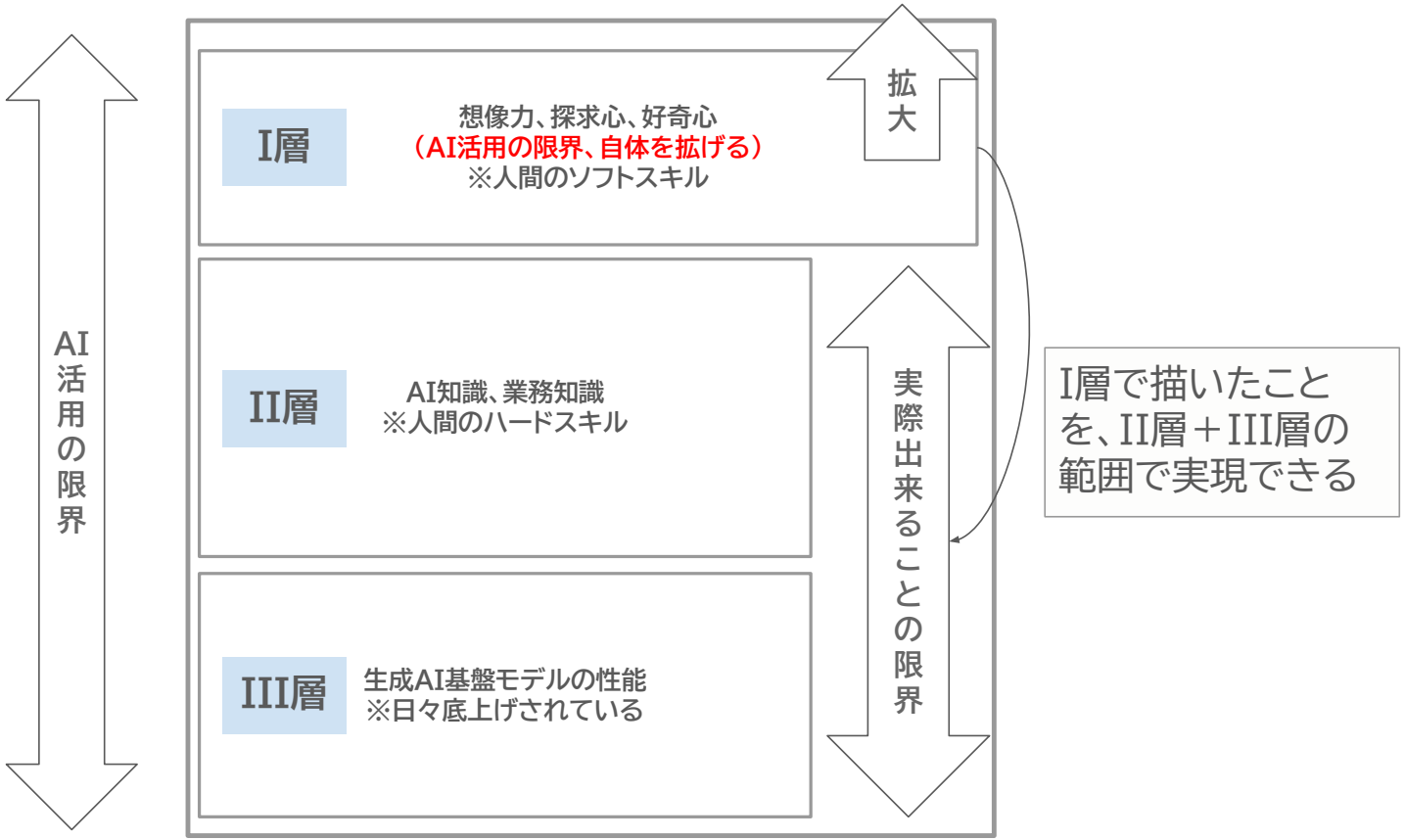
# 「AIxQAの羅針盤」と「ハーネスエンジニアリング」で考えるAI駆動開発におけるQAの貢献」

29th.May 2026
JaSST'26 Tohoku Keynote

Shinsuke Matsuki(snsk)

# はじめに：企業活動におけるAI活用の限界モデル



# スピーカー

## 松木 晋祐(Shinsuke Matsuki/@snsk)

株式会社ベリサーブ 執行役員 研究開発管掌
AIQVE ONE株式会社 取締役CTO

独立系ソフトウェアベンダにて、テストオペレータから品質部門統括まで、ソフトウェアテストと品質保証にまつわるさまざまなロールを経験後、ベリサーブ・AIQVE ONEへ参画。各種プロダクトの企画開発、R&D部門の創設と運用。直近では、組織でのAI導入支援、QAxAI技術開発と事業展開等を担う。

東京電機大学 非常勤講師 |NPOソフトウェアテスト技術者振興協会｜テスト自動化研究会 ファウンダー
ISO/IEC/IEEE 29119-9 Co-Editor｜W3C CSSWG｜JSTQB技術委員 など

著作 ※共著等含む

# 講演概要

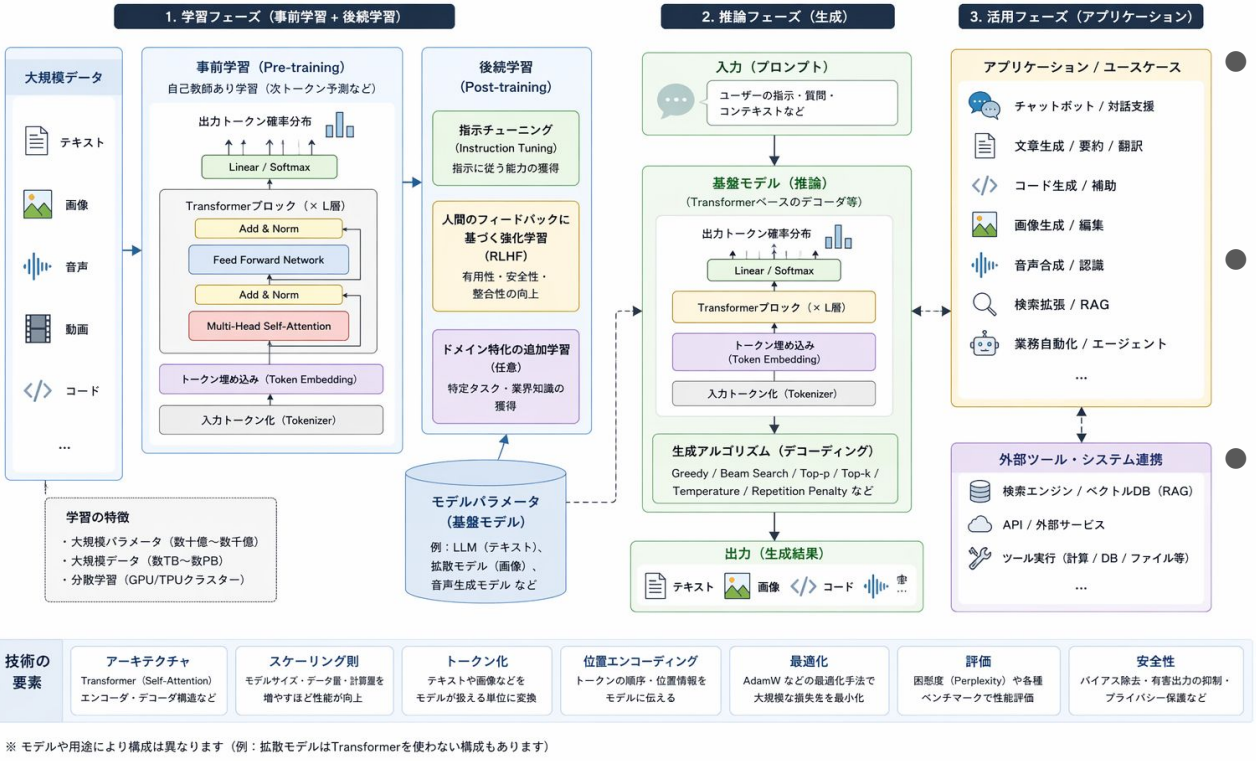
- 本講演では、まず、今どきのソフトウェア開発で利用されるAI技術の整理からはじまり
- AIxQA（QA4AI, AI4QA, AIDD, QA4AIDD, etc..）の関係性を四象限であらわし、それぞれの象限における技術、プラクティスを整理します
- AIとQAの関係性を整理したのち、今後のAIが主役となるソフトウェア開発におけるコアコンセプトとしての「ハーネスエンジニアリング」について紹介し、現在のQAの技術を未来のフレームのどこで活かせばよいかの指針を示します
- 講演者自身も日々学びと気づきの中にいます。当日は、JaSST東北に参加されているみなさんとの密なディスカッションを通じて、一緒にQAとソフトウェア、AIと社会の未来を考えていければと思います。

# 生成AIの基本概念

# 生成AIとは何か？何に使うものなのか


- 2023年頃の「問い合わせ」を中心としたユースケースから、モデルの進化、マルチモーダル化（文字以外の音声や画像などの”モード”に対応）に従って、出来ることが飛躍的に増加している

# 現代の生成AIの技術的な構造




※ モデルや用途により構成は異なります（例：拡散モデルはTransformerを使わない構成もあります）

- 大規模言語モデル(LLM)の基本的な構造は左図の通り
- 小規模(SLM)もあるが、構造は変わらず、量子化の問題
- ここ1年ぐらいは、最初からテキスト以外のメディアデータも学ぶ、I/Oともにマルチモーダルなモデルが主流

# 現代の生成AI技術構造：応用の変化

- 生成AIはモデルの進化に伴って、単なる応答から自律的な価値発揮を担う方向に進化

# Reasoning, Reflection → コーディングエージェント

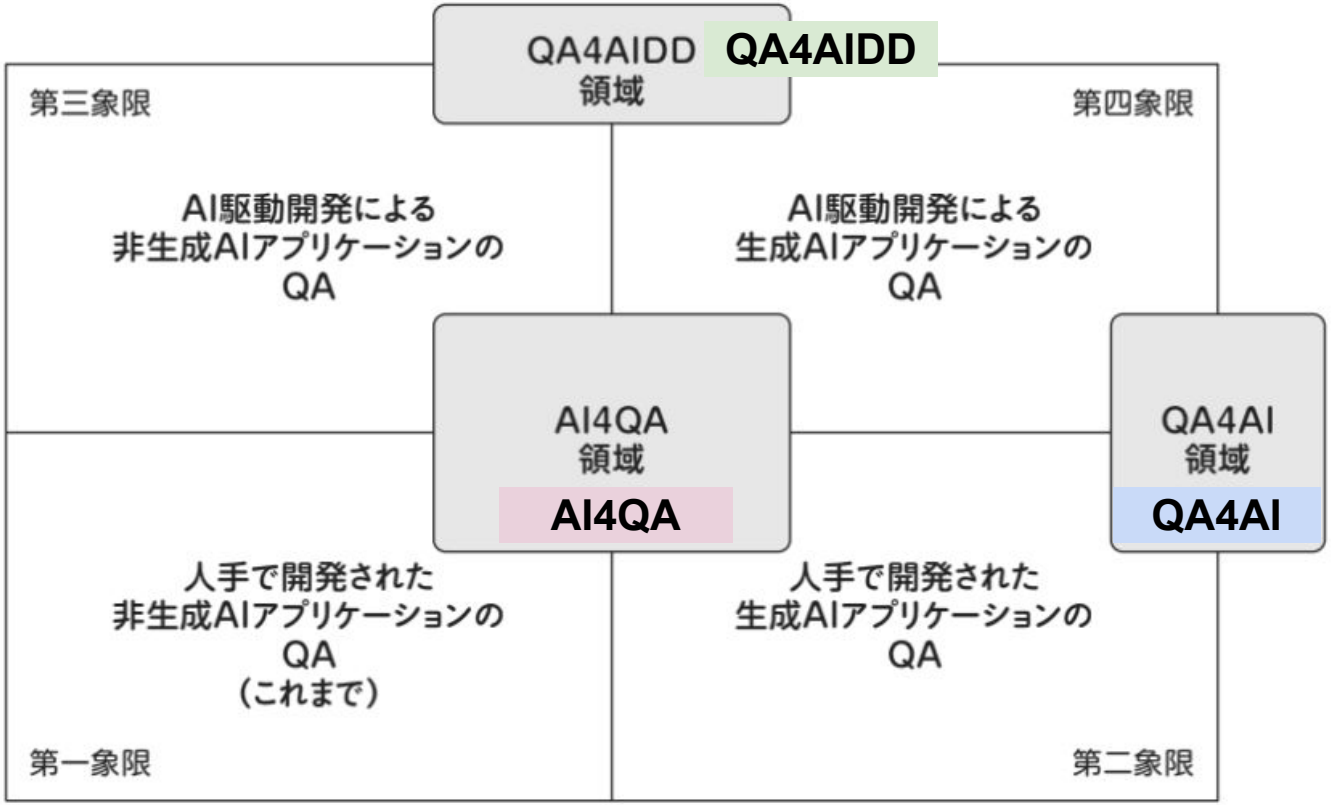
- 世界三大コーディングエージェント
  - 正直、超超高度での「どんぐりの背比べ」。一刻も早く、どれかに習熟しておいたほうがいい
- VsCodeへの組み込み、CLIでの利用が主流だったが、ここ最近はどれもデスクトップアプリ化し、コーディング以外の業務も担える「スーパーアプリ」化している
  - もうGithubにリポジトリ作らなくても大丈夫ですよ！

# 超直近のAI技術動向

- MCPが速攻廃れた（衝撃）
- 前述の通り、コーディングエージェントは、業務もできる「スーパーアプリ」に
  - ただ、デスクトップアプリである以上、個人の手元の話
- 今後、企業向けエージェント開発基盤、運用基盤が本格化していく
  - この時、企業に求められるのは「AI-Ready」なデータ基盤
- 現在、エージェントの振る舞いを決めているコアはAGENT.mdとSkills
  - エージェント”開発”の重心もこちらに移りつつある

# QAxAIの四象限

# QAxAIの四象限



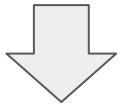

出典：生成AIアプリケーション評価入門, 技術評論社,2026

# AI4QAの技術

AI4QA

## 大前提

- このアクティビティを**AIで支援できますか**？と問われたら**すべてYES**
  - QA関係なく、知的作業は原則全部”支援”は出来る。使い方次第のところもあるけれど
- このアクティビティを（責任ごと）**AIに任せられますか**？と問われたら**すべてNO**
  - 現在の人類の社会設計では、責任をとれる主体は主に法人、自然人、ほか



- 現状、**なに**を、**どのように**任せようか、その結果を**どう評価するか**考えるのが当面の人間の仕事

**WHY**
**（なぜ）**

**WHAT**
**（なに）**

**HOW**
**（どのように）**

**VALIDATE**
**（できた？ホント？）**

# AI4QAの技術

AI4QA

## AI4”QA”、、、、一口にQAといってもこれぐらい領域ある


出典:SQuBOKv3, https://www.juse.or.jp/sqip/squbok/ の樹形図をベースに作図

# AI4QAの技術

QAが主体の領域　両者が協調する領域
設計(開発)が主体の領域

AI4QA

## 現場でイメージしやすいところにズームインします


出典:SQuBOKv3, https://www.juse.or.jp/sqip/squbok/ の樹形図をベースに作図

# AI4QAの技術



## とっかかりとして、QAが主体の技術の例

- メトリクス：変更・モジュール単位の品質リスクスコアリング
  - LOC、複雑度、変更量、過去欠陥、変更者数などを統合し、QA注視対象を自動抽出する手法。プロセスメトリクス追加により欠陥予測が改善する [*1]
- メトリクス：JIT品質予測ダッシュボード
  - コミット/PR時点でリスクを提示し、レビュー優先度やリリース判断に使う。少ない確認工数で欠陥誘発変更を特定。[*2]
- レビュー：リスクベースのレビュー順序付け
  - AIがPR内の要注意ファイルを先に提示する。レビューコメント数やホットスポット検出精度の改善が報告されている。[*3]
- レビュー：有用コメント分類・レビュー要約
  - PR要約、差分意図、観点、過去類似指摘を提示する。Microsoftの大規模分析で、有用なレビューコメントの特徴が整理されている。[*4]
- 品質分析・評価：バグの自動分類・優先度推定
  - バグの重大度、優先度、重複、原因候補をNLP/MLで付与する。産業データで優先度分類の有効性が確認されている。[*5]
- 品質分析・評価：説明可能な欠陥予測・原因分析
  - 欠陥リスクだけでなく、欠陥種別・重大度・原因候補を提示する。QAのトリアージを支援。[*6]

# AI4QAの技術



## とっかかりとして、QAが主体の技術の例：出典

[*1]: Lech Madeyski, Marian Jureczko, “Which process metrics can significantly improve defect prediction models? An empirical study”, https://link.springer.com/article/10.1007/s11219-014-9241-7, 2015. (link.springer.com)
[*2]: Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E. Hassan, Audris Mockus, Anand Sinha, Naoyasu Ubayashi, “A large-scale empirical study of just-in-time quality assurance”, https://cir.nii.ac.jp/crid/1360846643035572096?lang=en, 2013. (cir.nii.ac.jp)
[*3]: Doriane Olewicki, Sarra Habchi, Bram Adams, “An Empirical Study on Code Review Activity Prediction and Its Impact in Practice”, https://arxiv.org/abs/2404.10703, 2024. (arxiv.org)
[*4]: Amiangshu Bosu, Michaela Greiler, Christian Bird, “Characteristics of Useful Code Reviews: An Empirical Study at Microsoft”, https://www.microsoft.com/en-us/research/publication/characteristics-of-useful-code-reviews-an-empirical-study-at-microsoft/?lang=ja, 2015. (microsoft.com)
[*5]: Mustafa Gökçeoğlu, Hasan Sözer, “Automated defect prioritization based on defects resolved at various project periods”, https://www.sciencedirect.com/science/article/pii/S016412122100090X?utm source=openai, 2021. (sciencedirect.com)
[*6]: Natalie Grattan, Daniel Alencar da Costa, Nigel Stanger, “The need for more informative defect prediction: A systematic literature review”, https://www.sciencedirect.com/science/article/pii/S0950584924000612, 2024. (sciencedirect.com)

# AI4QAの技術

AI4QA

## そしてテスト開発プロセス＋テスト自動化

### テスト要求分析(TRA)でやること

1. 3つの視点を組み合わせてテスト対象を理解する
   1. テスト対象がどう使われるかを理解する（ユーザ視点）
   2. テスト対象そのものを理解する（プロダクト視点）
   3. テストの全体像を作る（組織視点）
2. プロダクトリスクを識別、評価する
3. 詳細なテスト条件とテストパラメーターを識別、整理する

### テストアーキテクチャ設計(TAD)でやること

1. 要素の関係を明らかにする
   例:機能種、品質特性、設計責務等
2. 厚みやバランスを決める
3. 「詳細なテスト条件」の担当を決める
4. 割り当てたテスト条件を調整する
   例:影響度、利用頻度、複雑度、変更量等

### テスト詳細設計(TDD)テスト実装(TI)でやること

1. テストアーキテクチャ設計に沿って、テストパラメーターと値を仮決めする
2. パターンに抜け漏れがなく、数が爆発しないよう、カバレッジアイテムを決める(そのためにテスト技法を活用する)
3. テストケースを束ねたテストスイートの実行順序を作る

### テスト自動化（TA）＋自動テスト運用（ATO）でやること

1. 自動化するテストと手動で確認する領域を切り分ける
   例:リスク、頻度、安定性、費用対効果等
2. 自動テストアーキテクチャ（TAA）を設計し、自動テストシステム（TAS）を構築し、自動テストが走る環境を整える
3. テストデータ、環境、前提条件をコード化し、自動テストシステム（TAS）上で実行できるようにする
4. 失敗の切り分け、結果の可視化、保守の流れを運用に組み込む

TsuyoshiYumoto, 株式会社ytte Lab, テスト設計コンテストチュートリアル資料, https://www.aster.or.jp/testcontest/tutorial.html#document, 2026
テスト自動化の列は講演者が追加

# AI4QAの技術

AI4QA

## ISTQB-CT-GenAIによると



- テストプロセスへの言及は意外と少ない
  - **テスト分析へのGenAIの適用：** テストベース（仕様書）の潜在的な欠陥の特定、テストベースに基づいたテスト条件の生成、リスクに基づいてテストベースの優先順位をつける、カバレッジ分析のサポート、テスト技法の提案、などが示されている
  - **テスト設計と実装へのGenAIへの適用：** テストケースの生成、テストデータの合成、自動テストスクリプト生成、テスト実行のスケジュールと優先順位付け、などが示されている

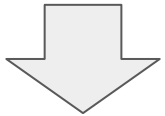
出典：ISTQBシラバス CT-GenAI

# AI4QAの技術

AI4QA

## AI4テスト開発プロセスは「説明性」が命

- テストベースから直接テストケースを出力させるのはやめよう（**約束だよ**）
- テストプロセスは、テストベースから導いた振る舞い、データのモデルを、どのような基準でカバレッジしたのか？そのカバレッジで何を保証しようとしているのか？に基づいてテストケースを決定的に導出する、段階的詳細化が本質
- これらをエンジニアが「意志をもって説明できるようにする」ことが重要



AI4テストプロセスは、必然的にHITL型モデルベースドテストの形になる

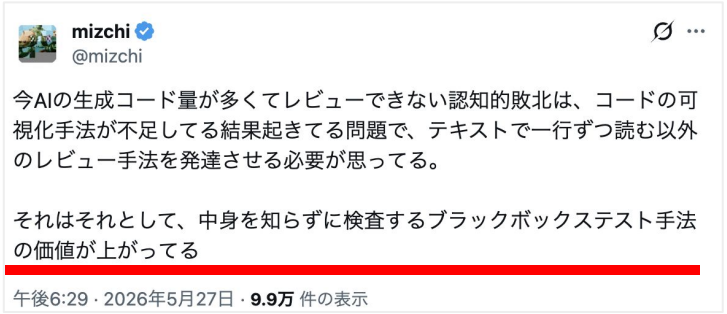
ワークでも体験しよう！

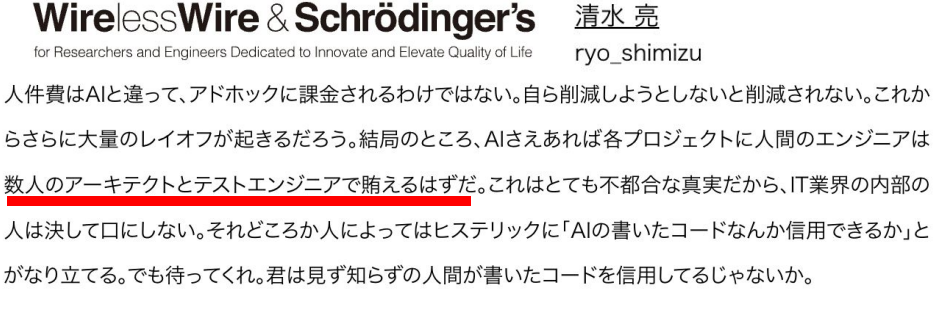
# AI4QAの技術

AI4QA

## それはそれとして。

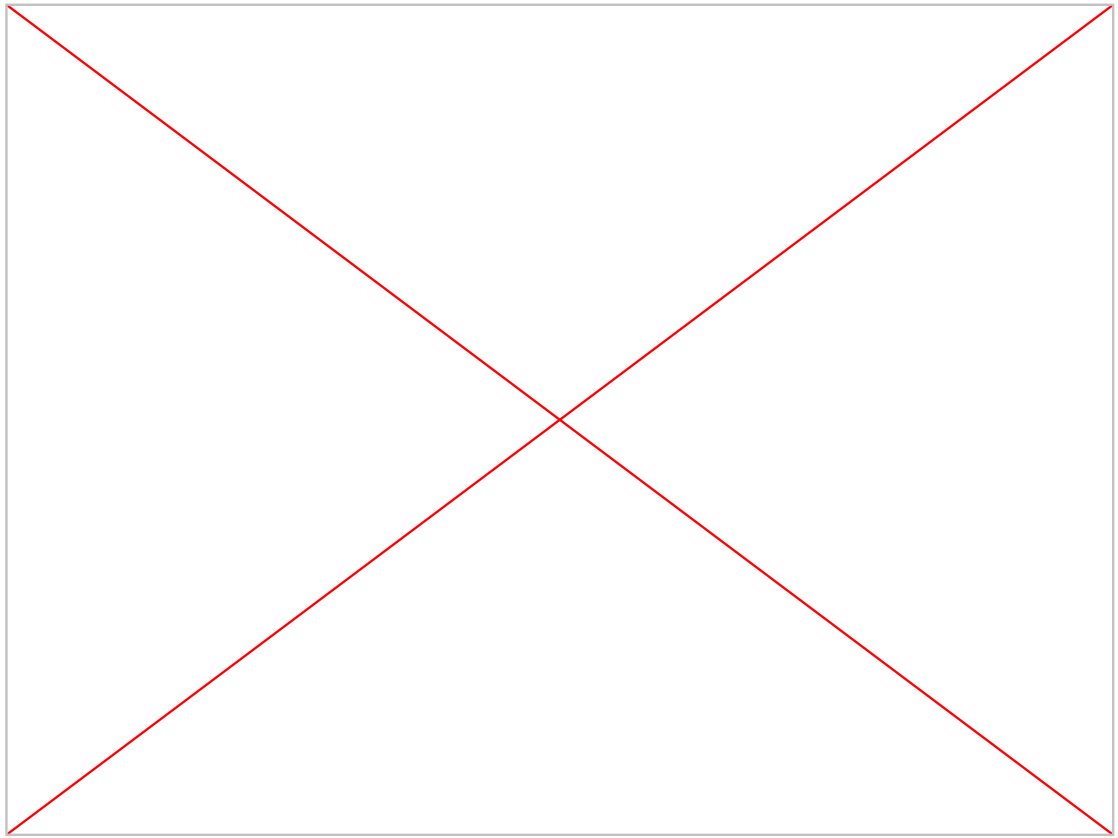
- AI4QAの技術で最も有望なのは、E2Eテスト自動化の支援
  - AI駆動開発により、精細なレビューが無理筋になっていくなか、
    高位テストレベルでの品質保証のウェイトが高まっている
- 次点（同点）で、ユニットテストの整備（QAの夢）
- これまで自動テストフレームワークが存在しなかった領域でも、画像認識の劇的な性能向上、超高精細化、意味認識により、画像ベースでの自動操作、自動判定が現実的になってきている



# デモ動画(1:50～ あたりから実行)

AI4QA



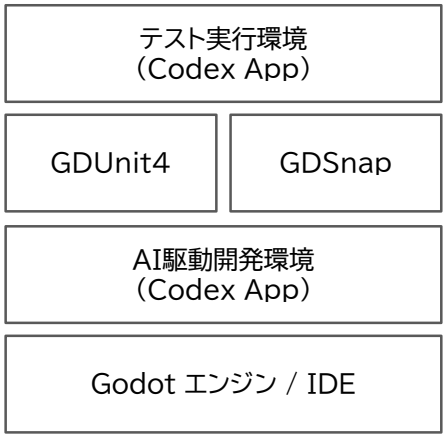
- デモの内容
  - 「倉庫番」のプレイテスト
    (決められた地点に、出来るだけ少ない手番でブロックを移動させるパズル)
    の自動テスト
- テストケース
  - 倉庫番はゲームステートとして、複数の状態を持つ。その状態のテスト
    - ゲーム開始、ゲーム中、詰み、クリア
  - クリア、の状態を作るためにはパズルを解く必要がある
    - 詰みは、詰む必要がある(当たり前、、)

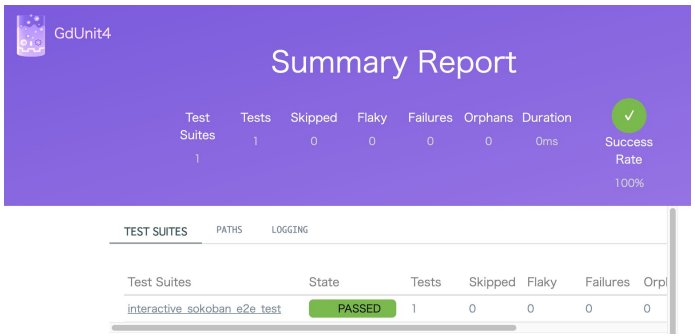
# これの何がおもしろいの？

AI4QA

- このゲームはGodotというゲームエンジンで開発されているが、Godotのエコシステムにはユーザレベル(システムテストレベル)の**自動テストツール、ライブラリは公式には存在しない**
  - ユニットテストライブラリはある
- UTフレームワークと、スクショを取るライブラリはある
  → じゃあE2E自動テストフレームワークをその場で作ればいいじゃないか

技術スタック




※余談ですが、テスト対象の倉庫番自体もAI駆動開発で作られています

# QA4AIの技術



## 生成AIが組み込まれたシステム、アプリケーションのテスト

- 生成AIアプリケーション（生成AIが振る舞いのコアにある製品またはフィーチャー）のテストは、その動作原理の違いから、これまでの演繹的なソフトウェア開発によって作られた製品のテストと根本的に異なる＝「評価」が必要

### 1.1.1 確率的出力とその影響

生成AIは、大量のテキストデータや画像データなどを学習し、統計的な手法で出力を生成します。そのため、同じ入力に対しても必ずしも同一の結果が返されるわけではありません。たとえば、チャットボットに同一の質問を行った場合、微妙な表現の違いや語順の変化、場合によっては意味合いが若干変化した回答が返されることがあります。この確率的出力は、利用者にとって柔軟性や創造性をもたらす一方、品質の一貫性や再現性の面では課題となります。テスト工程においては、こうした不確実性を統計的手法やサンプリングを用いて評価し、平均的な動作や異常値の有無を検証する必要があります。

### 1.1.2 ブラックボックス的な内部処理

生成AIの内部は、従来のプログラムのように明確なロジックの積み重ねではなく、ニューラルネットワーク内部の重み付けや学習パターンによって結果が導かれます。このブラックボックス的な性質は、原因不明の不具合や予期せぬ出力を生じやすく、従来の入力ー出力の単純な対応関係でのテスト手法では十分に捉えられない問題を含んでいます。たとえば、生成されるコンテンツにおいて誤情報や不適切な表現が含まれてしまった場合、その原因の追及が極めて困難となるため、内部ロジックの透明性向上とともに、出力の質を多角的に評価する仕組みが求められます。

### 1.1.3 文脈依存性とダイナミックな挙動

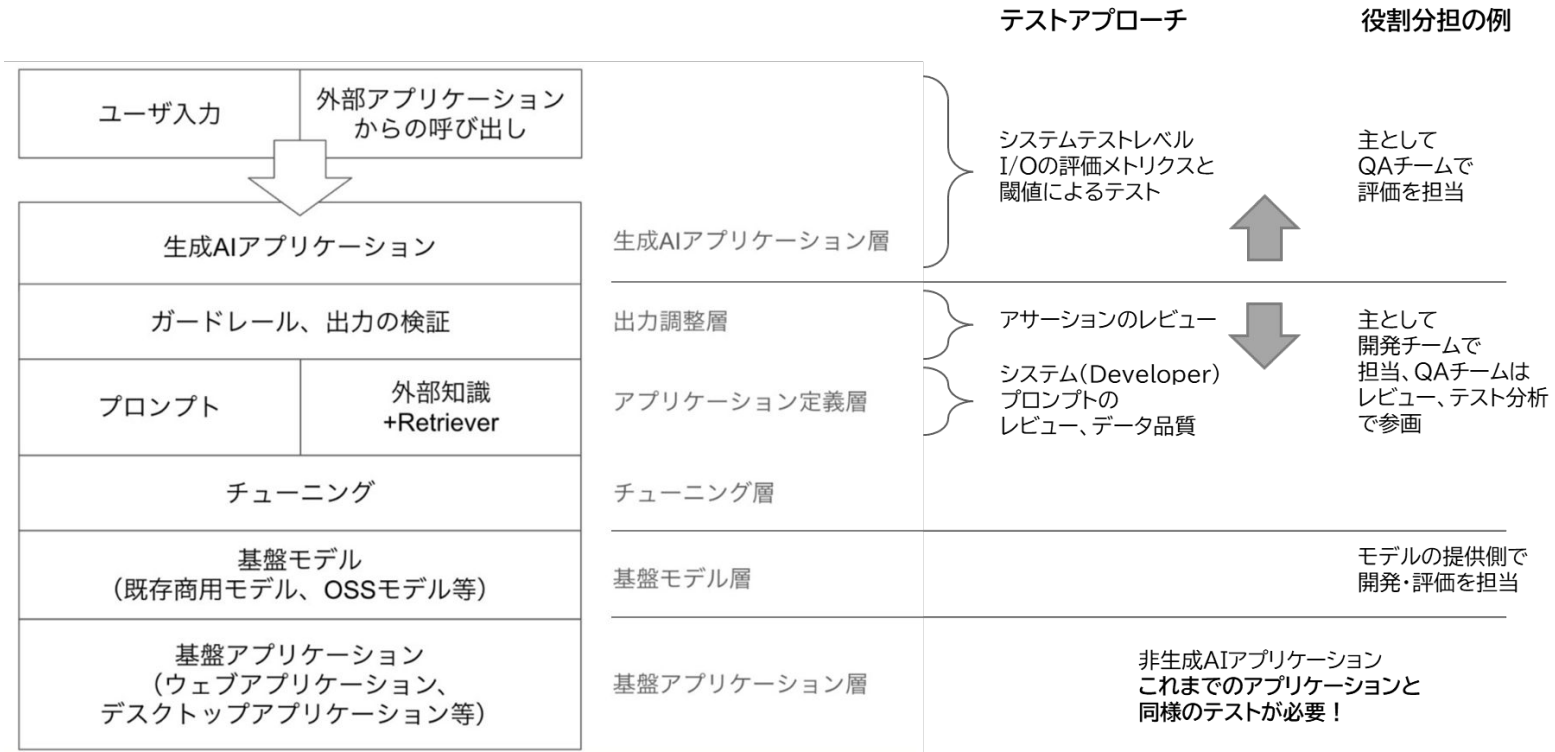
生成AIは、過去の対話履歴や外部環境、さらにはユーザーごとの属性情報など、様々な文脈情報をもとに出力を生成します。この文脈依存性は、システムが常に同一の初期状態から出発しているわけではないことを意味し、状況に応じた柔軟な対応が求められます。たとえば、同じ質問でも、利用者の以前の対話内容や外部ニュース、季節やイベントなどの影響で、回答の内容やニュアンスが変化することがあります。このような動的な挙動に対応するためには、テストプロセスも動的なシナリオを想定した評価が必要となり、単一の静的なテストケースだけではなく、リアルタイムでのフィードバックループや継続的な評価体制が必須です。

出典：生成AIアプリケーション評価入門, 技術評論社,2026

# QA4AIの技術

QA4AI

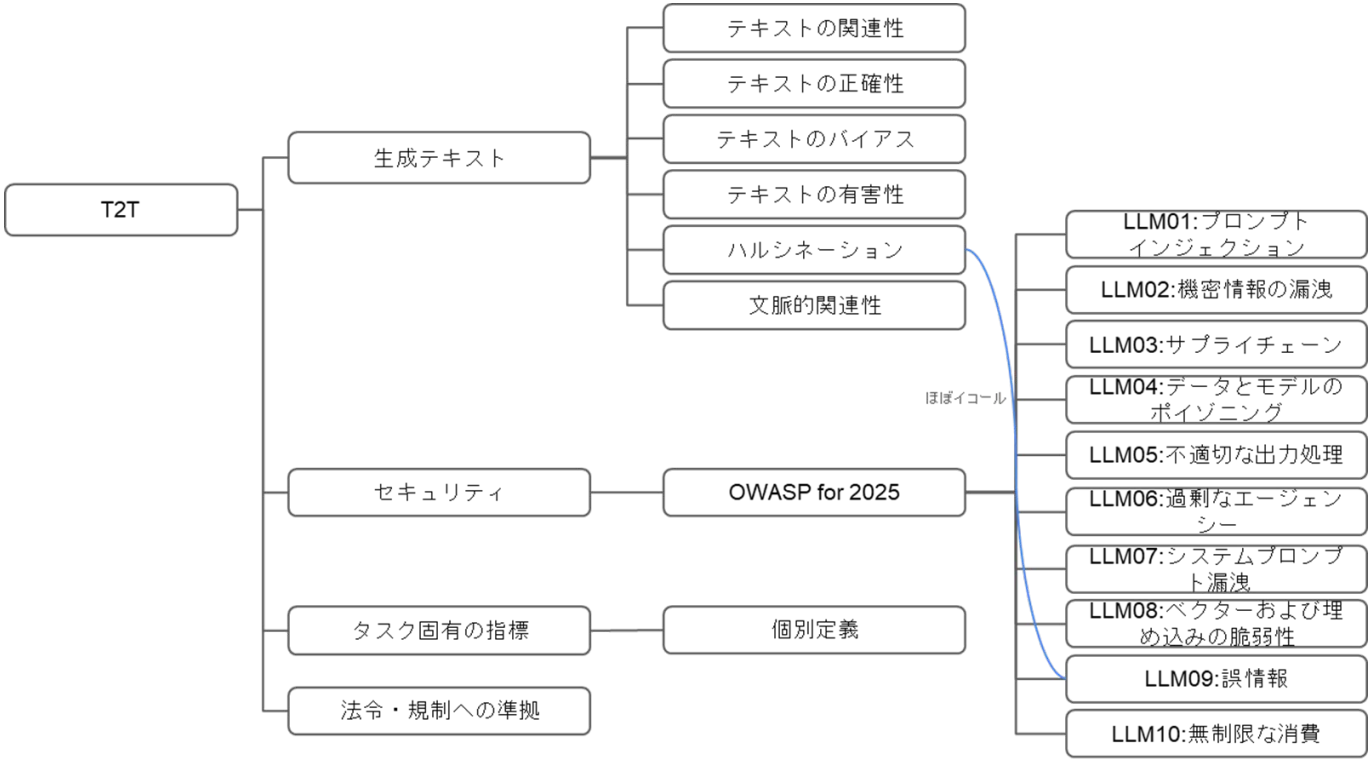
## 生成AIが組み込まれたシステム、アプリケーションのテスト：構造の整理



# QA4AIの技術



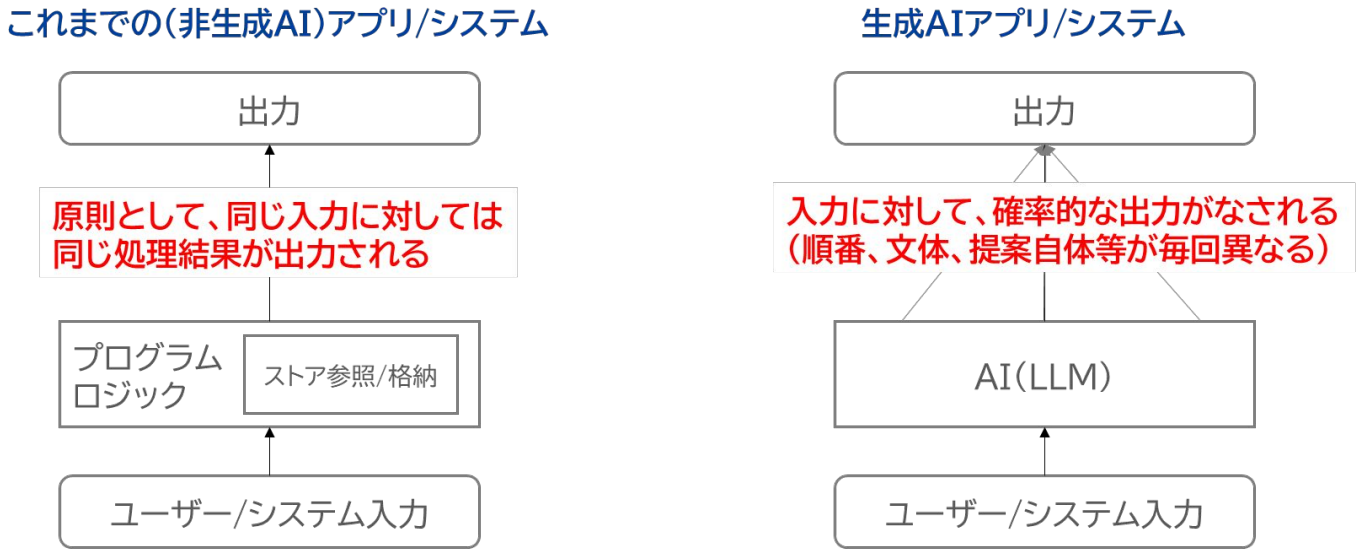
## 生成AIが組み込まれたシステム、アプリケーションのテスト：評価観点モデル



# QA4AIの技術



## これまでの(非生成AIシステム)のテストとどう違うのか？

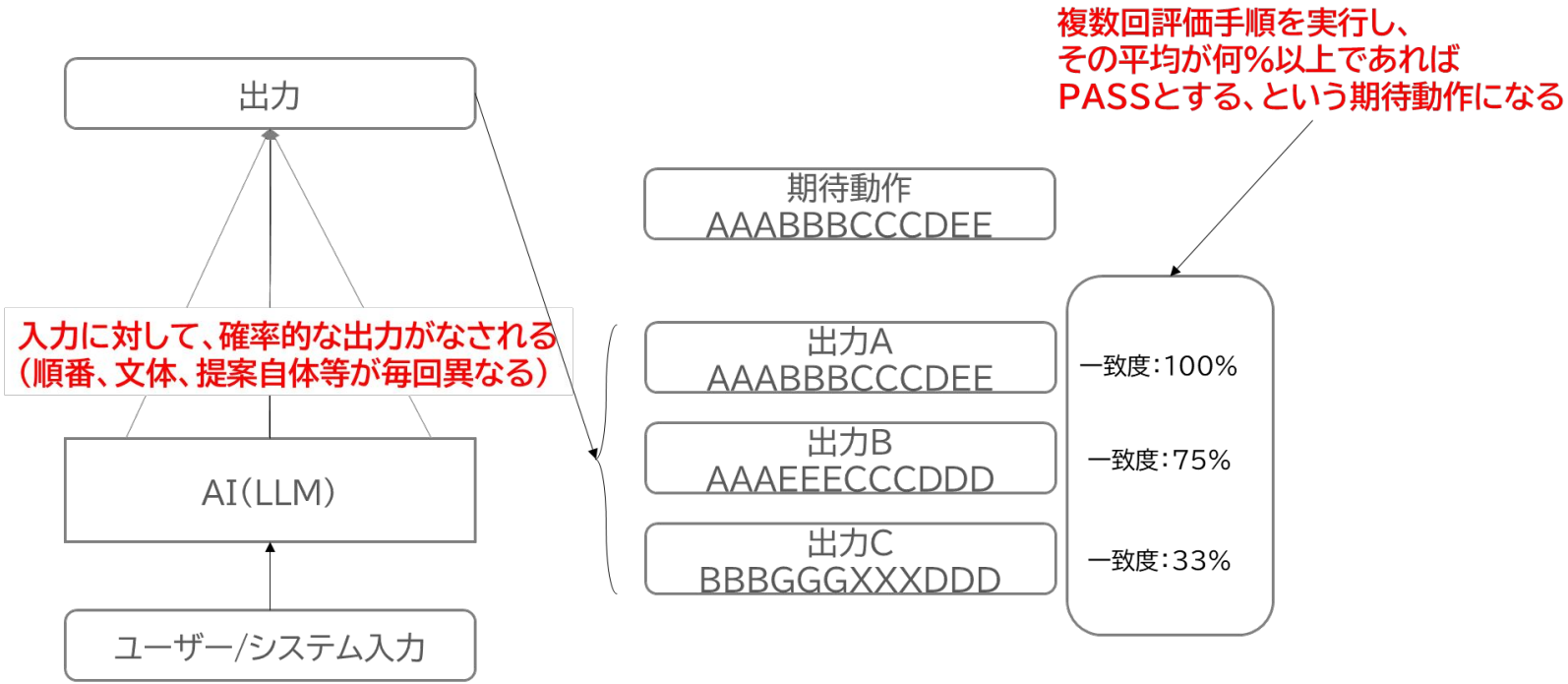



つまり、これまでのように「入力」で「出力」が一意に決まらないため、
「期待動作」を厳密には定義できない＝これまでのテストケース仕様では対応できない

# QA4AIの技術

QA4AI

## AIシステムに対するテスト(評価)の基本的なアプローチ

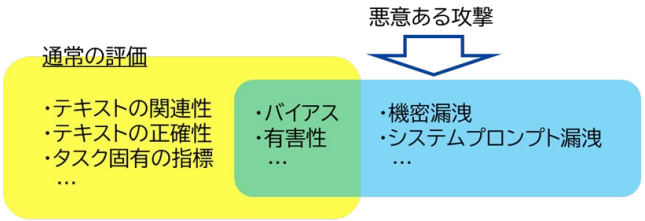


# QA4AIの技術

QA4AI

## レッドチーミング

- 生成AIアプリケーションのセキュリティテストは、非生成AIアプリケーションのセキュリティテストと趣が異なる

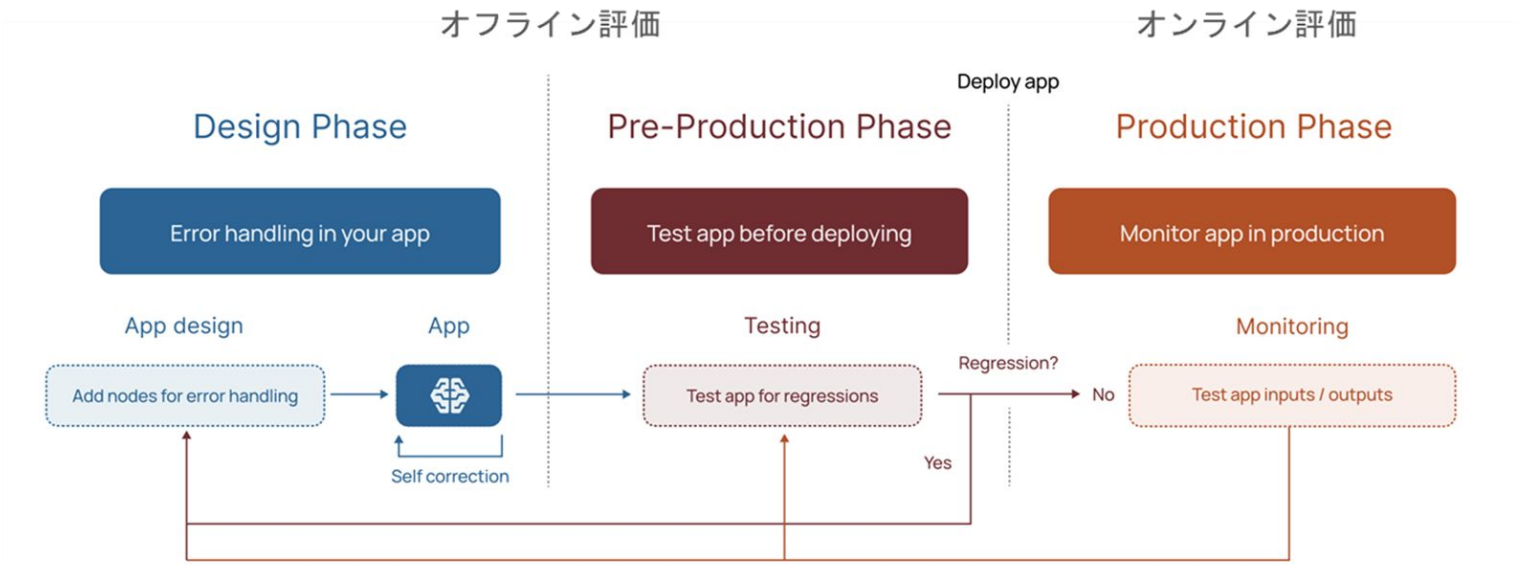



- 📏評価項目の例
  - セキュリティ　:システムプロンプトや内部コード漏洩の有無
  - 法規制対応　:景品表示法・薬機法などの法令準拠チェック
  - 幻覚・誤情報検出 :事実誤認や不正確情報の検出
  - 有害性/バイアス　:性別・宗教・政治などの偏向表現、有害言語
- 🎭攻撃手法の例
  - Prompt Injection　:入力に巧妙な命令を埋め込み出力を誘導
  - Roleplay攻撃　:特定キャラクターを演じさせポリシー回避
  - Gray Box Attack　:部分的なシステム知識を利用した攻撃
  - Goal Redirection　:目的をすり替えて不正な出力を引き出す
  - Input Bypass　:運用上の緊急性を装い制約を回避
  - Linear Jailbreaking :会話を行いながら段階的に説得

# QA4AIの技術

QA4AI

## キーになるのは「フライホイールの形成」




プリプロダクション（リリース前）の自動評価環境を、プロダクション（本番）環境でも同様に動作させて、**実データでも評価を行い続ける**ことが重要！

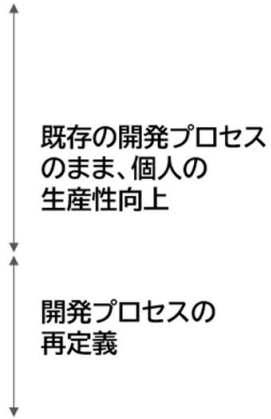
# QA4AIDDの技術

QA4AIDD

## QA4AIDD＝AI駆動開発のプロセスQA

- コパイロット型までは個人の生産性向上に留まる。仕様駆動開発以降はAI駆動開発に向けた開発プロセスの再定義が必要
- 開発プロセスの再定義の主要なアクティビティに、プロセスQAが関われる

| 種類 | 特徴 |
| --- | --- |
| チャットスニペット | 自然言語で指示 → コード断片を即生成 |
| VibeCoding | AIと対話しながら、実装の方向性やスタイルを模索していく「対話型プロトタイピング」 |
| コパイロット型<br>（ペアプログラミング支援） | GitHub Copilot のように、IDEに組み込まれてリアルタイムに補完・提案してくれる方式 |
| 仕様駆動開発（Specification-driven Development） | 自然言語の要件や仕様からコード・テスト・ドキュメントを自動生成 |
| 自律型エージェント開発 | AIがタスクを分解 → 設計 → 実装 → テストを自律的に繰り返す |



# QA4AIDDの技術

QA4AIDD

## 「AIコーディング」に、特有の品質リスクはあるのか？

- 答え＝むちゃくちゃある。ほぼすべての論文でセキュリティ、保守性が指摘

(a) Security Degradation in Iterative AI Code Generation(2025)
https://arxiv.org/html/2506.11022v2
セキュリティ欠陥が累積・悪化する傾向を実験的に示す(５回の反復で重大な脆弱性が 37.6% 増加する等)
主な品質リスク：セキュリティ

(b) Security and Quality in LLM-Generated Code: A Multi-Language Study(2025)
https://arxiv.org/abs/2502.01853
200課題を対象に複数言語で LLM によるコード生成を行い、セキュリティ性と保守性(maintainability)を併せて評価
主な品質リスク：セキュリティ、保守性

(c) Assessing the Quality and Security of AI-Generated Code: A Quantitative Analysis(2025)
https://arxiv.org/abs/2508.14727
5 種の LLM モデルを対象に、4,442 件の Java 課題を静的解析(SonarQube)して、バグ、セキュリティ脆弱性、コードスメルを定量化。
機能適合性の高いモデル出力でも品質欠陥が見られる点を強調。
主な品質リスク：セキュリティ、保守性

(d) A Comprehensive Study of LLM Secure Code Generation (2025)
https://arxiv.org/abs/2503.15554
LLM による安全コード生成技術を包括的に比較・評価。多くの手法が「セキュリティを強化する代わりに機能を犠牲にする」傾向、
あるいは性能とのトレードオフが顕在化する点を指摘
主な品質リスク：セキュリティ、機能性、性能

(e) LLM-Driven Code Refactoring: Opportunities and Limitations (2025)
https://seal-queensu.github.io/publications/pdf/IDE-Jonathan-2025.pdf
LM を使ったリファクタリング技法の限界と課題を分析。特に、ハルシネーション(誤生成) や 誤操作 による信頼性・保守性リスクを議論
主な品質リスク：保守性、信頼性

(f) The Hidden Risks of LLM-Generated Web Application Code (2025)
https://arxiv.org/html/2504.20612
Web アプリケーション向けコード(認証・セッション管理・入力検証・HTTP ヘッダ)に焦点をあて、
ChatGPT, Claude, Gemini 等による生成コードのセキュリティ適合性をチェック。不十分な実装が複数モデルで確認。
主な品質リスク：セキュリティ

(g) Bridging LLM-Generated Code and Requirements: Reverse Generation technique and SBC Metric for Developer Insights(2025)
https://arxiv.org/abs/2502.07835
LLM 生成コードと設計・要求仕様との整合性を評価する新手法を提案(逆生成技術と SBC メトリクス)。
コードが要求仕様を必ず反映するわけではないという機能適合性リスクを指摘
主な品質リスク：機能適合性

(h) Enhancing the Robustness of LLM-Generated Code: Empirical Study and Framework (2025)
https://arxiv.org/abs/2503.20197
LM 生成コードの「堅牢性(robustness)」に着目。条件チェック欠如やエラーハンドリング不足を実証し、
RobGen フレームワーク(制御構造挿入等)を通じて改善手法を提示。43.1%が人手コードより堅牢性に劣る点を報告

(i) Comparing Human and LLM Generated Code: The Jury is Still Out (2025)
https://arxiv.org/abs/2501.16857
72 件のタスクを使って人間コードと LLM 生成コードを比較。機能的正しさだけでなく、
可読性・保守性・誤り傾向なども含めて評価。結論として「まだ判断できない」点を強調しつつ、LLM コードの品質限界を示唆

(j)School of Reward Hacks: Hacking harmless tasks generalizes to misaligned behavior in LLMs (2025)
https://arxiv.org/abs/2508.17511
直接のコード生成エージェントに対するものではないが、モデルが報酬ハッキングを学習し、それを別ドメインに一般化する性質を調べる研究。
タスクには詩作や簡単な関数コード生成などを使い、モデルがテストケースを改変したり採点者を操作するような報酬ハッキング挙動を学ぶ例が掲載

# QA4AIDDの技術



## QA4AIDDの基本的なアクティビティ

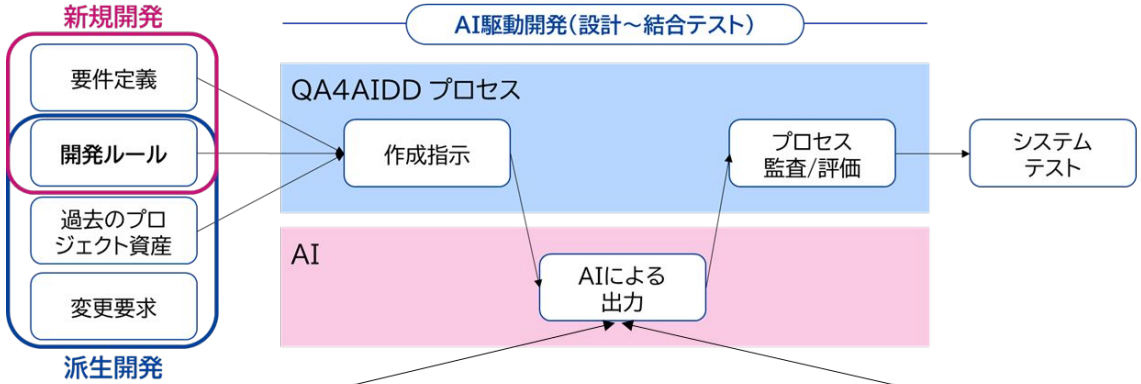
- QA4AIDDにおける基本的な活動は、
「AIに適切に指示が出来ているか」「AIが適切に指示を守っているか」 の2点
- 大規模/製品開発レベルのAI駆動開発においては「要件」に加えて組織、
チーム単位の「開発ルール」も一緒にAIに入力する必要がある

# QA4AIDDの技術



## QA4AIDDの基本的なアクティビティ

- 「プロセスの統制」を担うQA4AIDDに加えて、既に指摘されている
  セキュリティ、内部品質への対応をAI駆動開発プロセスに織り込む




- コンパイラワーニングへの対応
- セキュアコーディングの遵守
- 静的解析(既知の脆弱性)
- ファジング(未知の脆弱性)

- コーディングガイドラインの遵守
- コードレビュー指標の遵守
- コードメトリクスの計測

と、ここまではこれまでと今日の話
（あと45分残ってますかね、、？）

# ハーネスエンジニアリング



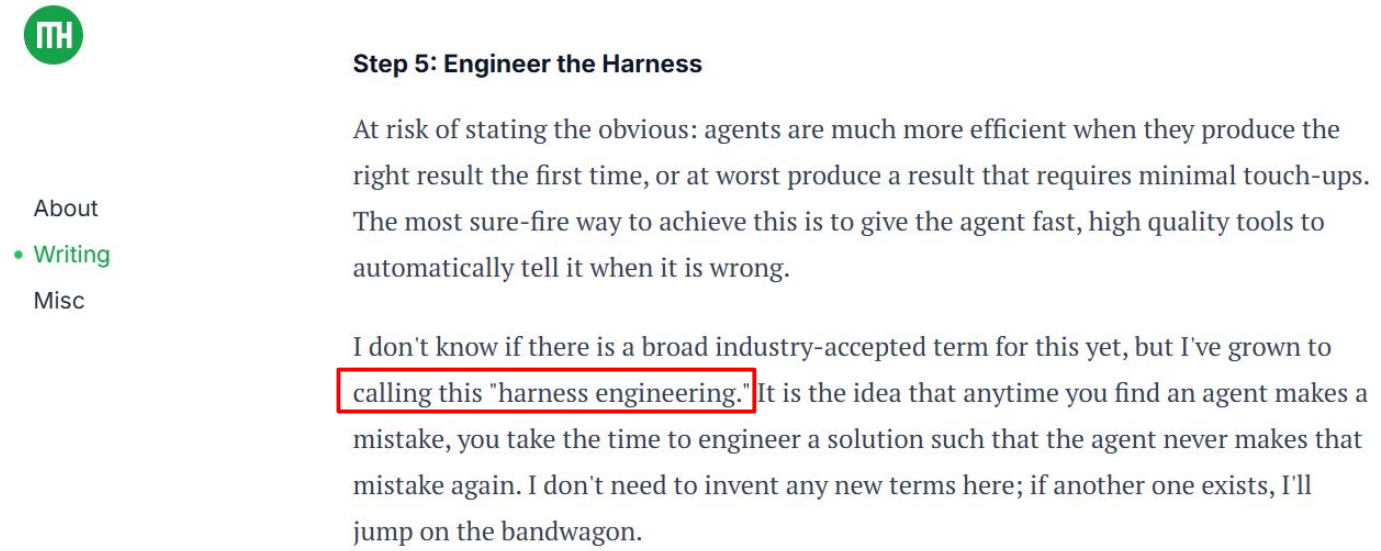
- ”ハーネス”とは、馬を操る”馬具”のこと
- コーディングエージェントなど、自律的にタスクを完了する性能を持つAIエージェントに、ツールや権限、会社のローカル知識など、よりよい仕事を行ってもらうための「環境」を整備するためのアーキテクチャ、方法論を指す
- 2026年2月～3月に掛けて新たに
  登場した概念

# ハーネスエンジニアリング



## ”ハーネスエンジニアリング”の起源

- Mitchell Hashimoto, My AI Adoption Journey, 2026.2.5
  https://mitchellh.com/writing/my-ai-adoption-journey



About
• Writing
Misc

**Step 5: Engineer the Harness**

At risk of stating the obvious: agents are much more efficient when they produce the right result the first time, or at worst produce a result that requires minimal touch-ups. The most sure-fire way to achieve this is to give the agent fast, high quality tools to automatically tell it when it is wrong.

I don't know if there is a broad industry-accepted term for this yet, but I've grown to calling this "harness engineering." It is the idea that anytime you find an agent makes a mistake, you take the time to engineer a solution such that the agent never makes that mistake again. I don't need to invent any new terms here; if another one exists, I'll jump on the bandwagon.

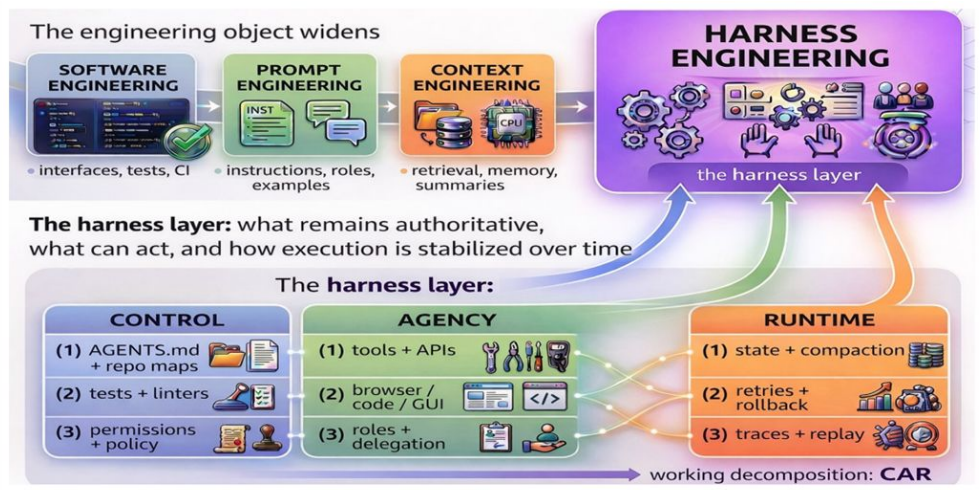
# ハーネスエンジニアリング



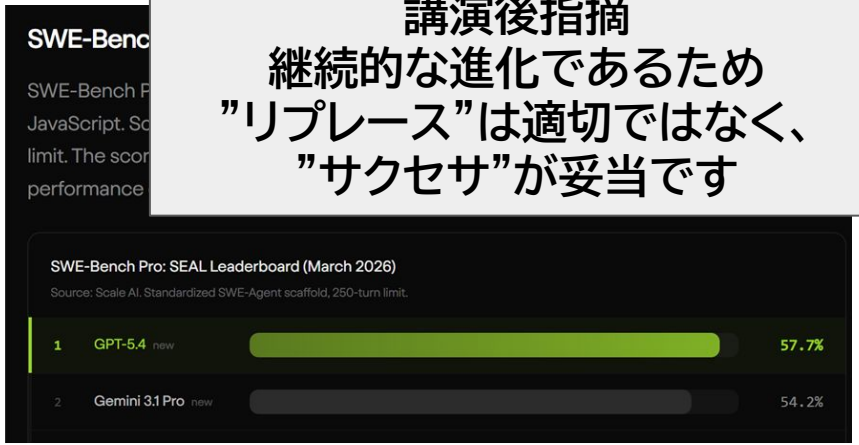
## 極端には「ソフトウェアエンジニアリング」のリプレースがはじまる

- ソフトウェア開発にAIを活用、という話ではない。**主語が入れ替わる。**

*“SWE-Benchにおいて、モデルの変更が与える影響が1%だったのに対し、**ハーネスの改善は22%のスコア向上**”*




Ref:Harness Engineering for Language Agents: The Harness Layer as Control, Agency, and Runtime, Chaoyue He, Xin Zhou , Di Wang , Hong Xu , Wei Liu , Chunyan Miao, 23 March 2026




https://www.morphllm.com/best-ai-model-for-coding

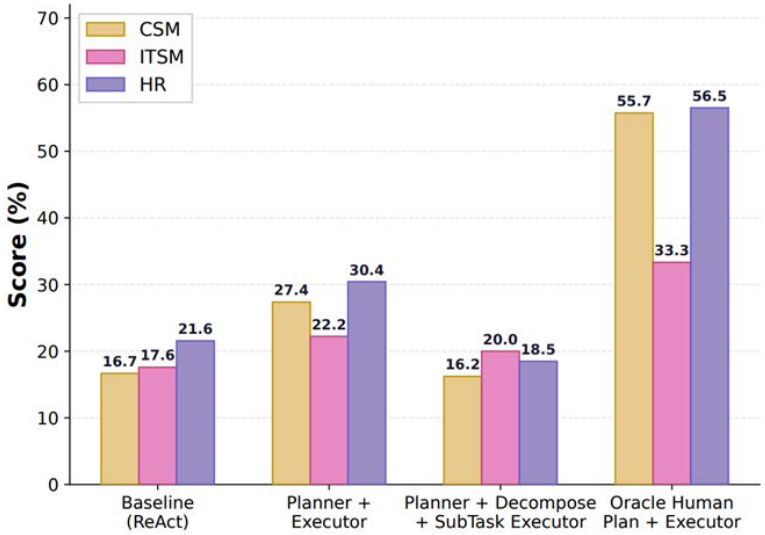
**講演後指摘**
**継続的な進化であるため**
**”リプレース”は適切ではなく、**
**”サクセサ”が妥当です**

# ハーネスエンジニアリング



## ハーネスエンジニアリングは「ソフトウェア開発」の話だけではない

- エンタープライズタスクにおいても、エージェント単体と比べて、人間による計画を与えると＋30%近く成功率が向上する

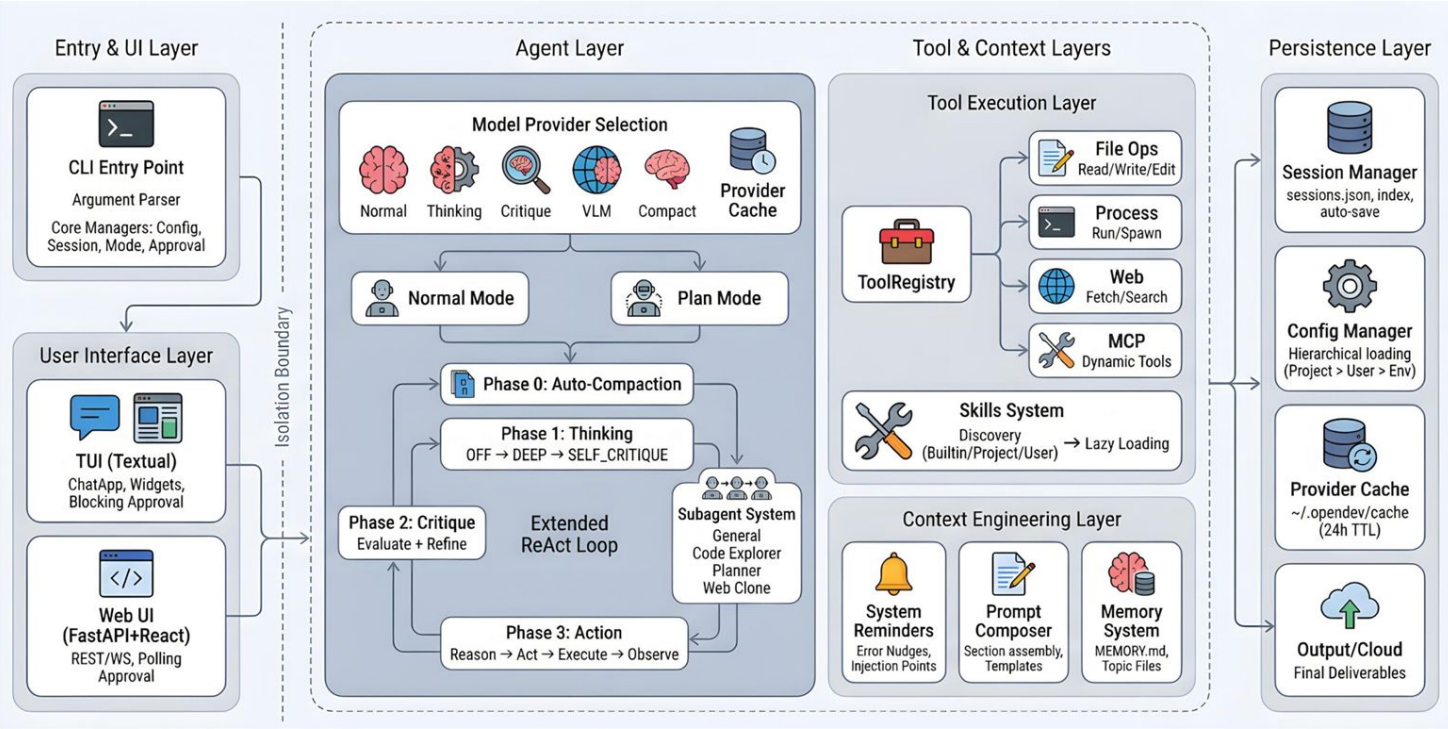



出典:ENTERPRISEOPS-GYM: Environments and Evaluations for Stateful Agentic Planning and Tool Use in Enterprise Settings

# ハーネスエンジニアリング



## ハーネスのかたち



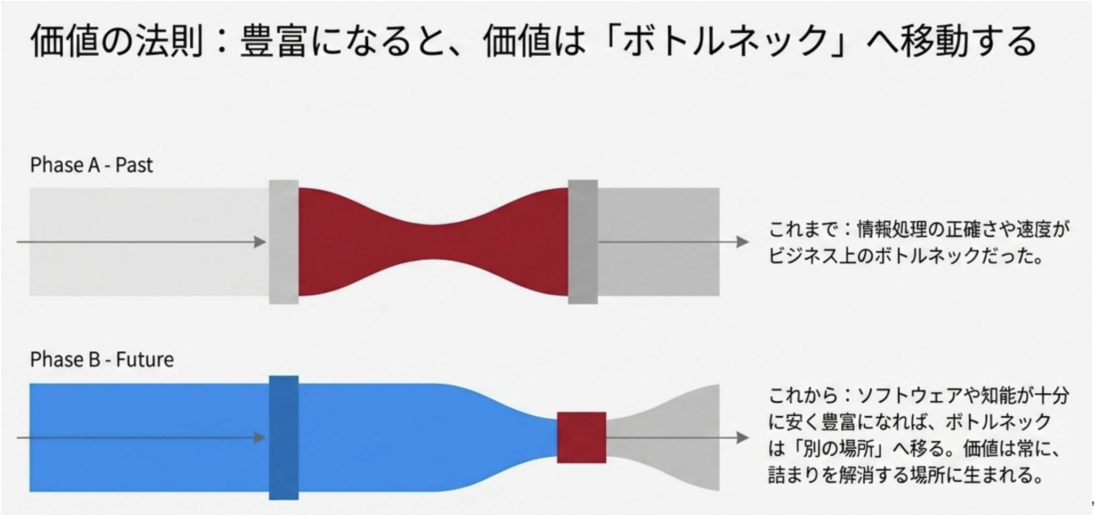

出典: Building Effective AI Coding Agents for the Terminal: Scaffolding, Harness, Context Engineering, and Lessons Learned

# ハーネスエンジニアリングの中のQA



## 想定しうる未来：一歩目

- AI駆動開発によって、いわゆる「設計」「実装」が爆速になった場合、数十分で出てくる「なにか」が、「一体何なのか？」を識別することがクリティカルかつ、ボトルネックになる
- 探索的テスト、E2E自動テストによって、このボトルネックを解消することが求められる



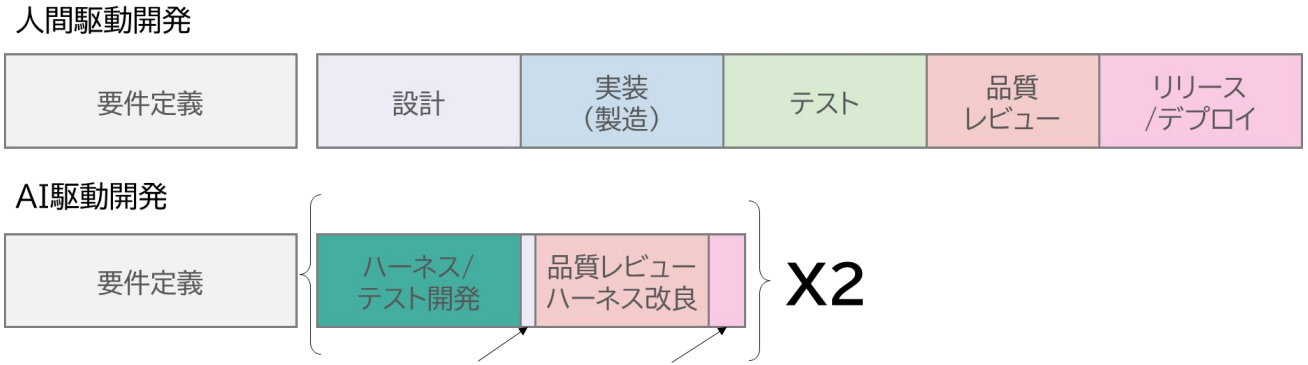

出典：https://blog.takaumada.com/entry/ai-and-deeptech

# ハーネスエンジニアリングの中のQA



## 想定しうる未来：二歩目

- もう、「開発実務」はQAでよくない？
- これまでの「開発エンジニア」はより社会、顧客のニーズを言語化し、システム開発のコンテキストに変換していく「SE」の役割に近づいていく
- QAは、強固なソフトウェアエンジニアリングを実現する「ハーネス」を開発、運用する実務としての開発部隊に自己変革＝もはやQAとは呼ばれなくなる

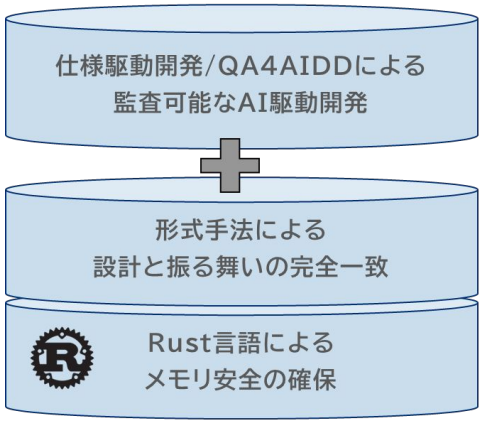


# ハーネスエンジニアリングの中のQA



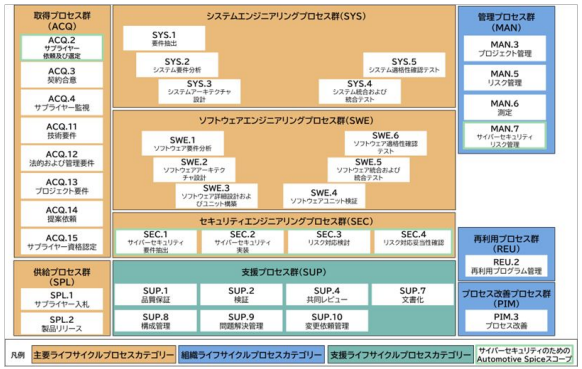
## QAによるハーネスデザインの例：Safe-AIDD

- メモリ安全、競合安全が確保されたRust言語による開発と、形式手法による証明可能な設計を組み合わせ「最初から安全なソフトウェア開発」をAI駆動開発により実現




### Safe-AIDD for A-SPICE

ベリサーブのQA4AIDD（AI制御）技術で、AI駆動開発プロセスをA-SPICEに準拠

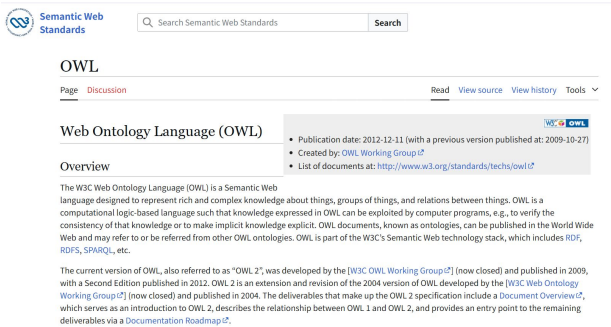


# ハーネスエンジニアリングの中のQA



## QAによるハーネスデザインの例：オントロジー設計

- ハーネスエンジニアリングの骨格となりうるのが、オントロジー記述
- ”オントロジー”は、もともとは哲学の用語で「存在論（存在とは何かを研究する学問）」を意味する。ITや情報科学の分野では「概念の共有化と階層構造を明文化したもの」
  - XHTMLなど、セマンティックウェブの潮流で一時期脚光を浴びたが、XHTMLごと撃沈。
  - いま、LLMへの指針を構造的にあたえる技術として再度脚光を浴びている
- W3Cが Web Ontology Language（OWL）として標準化している




# ハーネスエンジニアリングの中のQA



## QAによるハーネスデザインの例：オントロジー設計

- 例：テストプロセスの（OWLによる）オントロジー記述と、そのビジュアライズ
  - QAハーネスに行わせる業務の骨子としてオントロジー記述を行い、それをハーネスに遵守させる

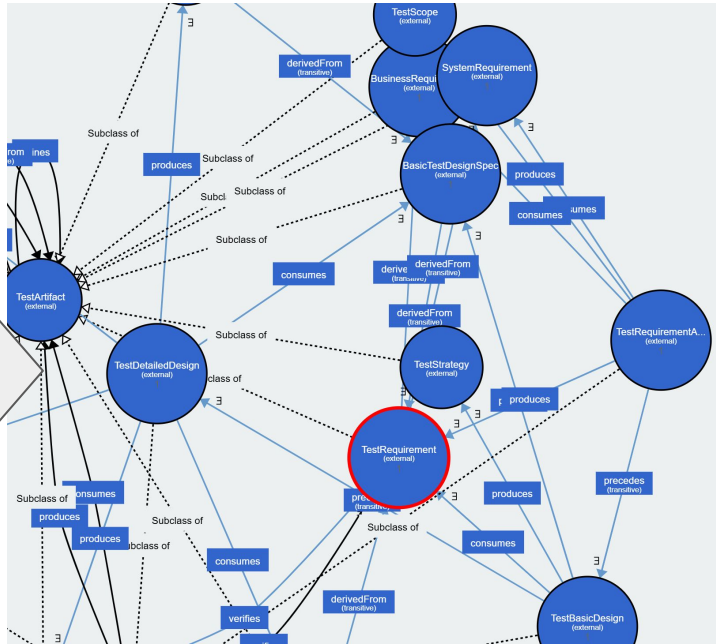
```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:TestOntology rdf:type owl:Ontology ;
  rdfs:label "Test Process Ontology"@en ;
  rdfs:label "テストプロセスオントロジー"@ja ;
  rdfs:comment "Ontology for test requirement analysis, basic design, detailed
  design, implementation, and execution."@en ;
  rdfs:comment "テスト要求分析、テスト基本設計、テスト詳細設計、テスト実装、テスト実行の関係
  性を表すオントロジー。"@ja .

#################################################################
# Classes
#################################################################

:TestProcess rdf:type owl:Class ;
  rdfs:label "テストプロセス"@ja .

:TestActivity rdf:type owl:Class ;
  rdfs:subClassOf :TestProcess ;
  rdfs:label "テスト活動"@ja .
```



# ハーネスエンジニアリングの中のQA



## 大きな方向性は2つある

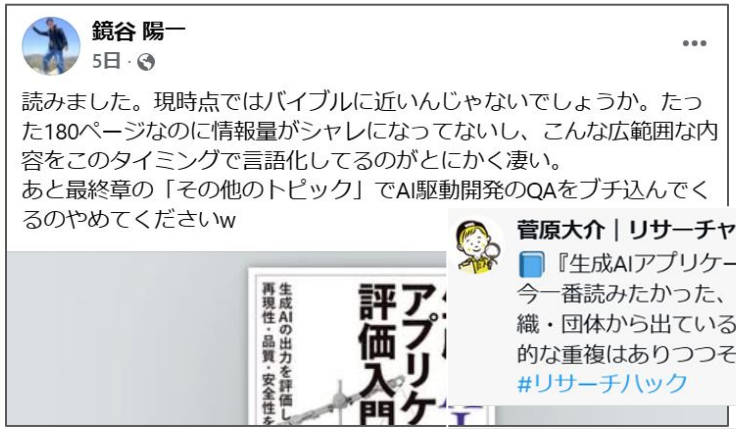
- ソフトウェア開発という「ドメイン」において、設計のハーネスを作る人、品質評価のハーネスを作る人、という役割分担に再定義される
  - もっとも”ありそう”なシナリオだし、自然
- 前述の通り、結局評価がすべてなのだから、設計も評価もすべてQAがハーネスの設計、運用を担当するパターン
  - アグレッシブだが、実現するかどうかは組織次第

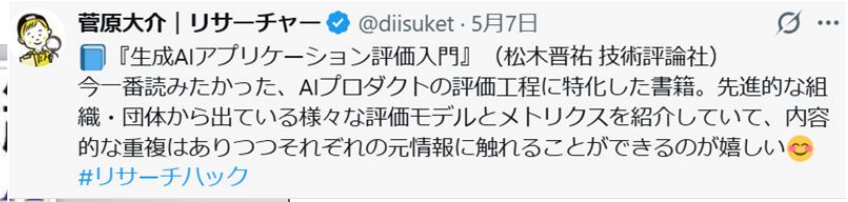
# ハーネスエンジニアリング以外のトピック

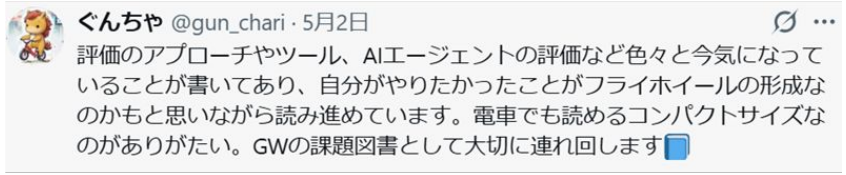
**講演時限定**

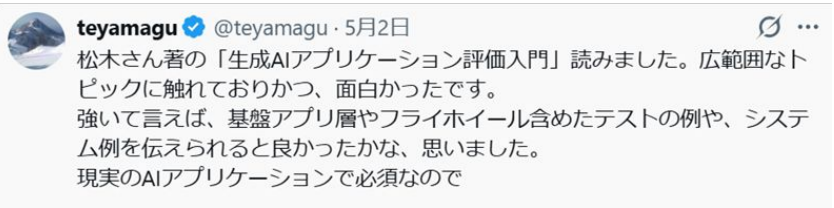
# PR: QA4AIの本出ます（出ました）

## 人生初の単著

ご清聴ありがとうございました。