# アカウンタビリティを軸とした AI-Native 開発のデザイン戦略

株式会社NTTデータグループ
〒135-6033
東京都江東区豊洲3-3-3豊洲センタービル
Tel: 03-5546-8051 Fax: 03-5546-2405
https://www.nttdata.com/jp/ja/

# Contents

# 要約

コーディングエージェントを開発の主体に据える AI-Native ソフトウェア開発は、アカウンタビリティの要求水準から逆算してプロセスの自律性と透明性を設計すべきである。コーディングエージェントが「動くもの」を高速に生成できる時代において、商用システムの開発では動くこと（機能性）だけでなく、期待通りに動くこと（品質）、開発プロセスが透明であること（透明性）、意思決定の根拠を説明し正当化できること（アカウンタビリティ）が求められる。

本稿では、スリーラインモデルをソフトウェア開発に適用してガバナンス構造を整理し、AI が担う範囲が段階的に拡大する3つの成熟度レベルを提示する。Level 1（AI-Generated）では AI が開発実務を担い人間が品質を作り込む段階を、Level 2（AI-Verified）では AI が品質の監督も担い人間がプロセス管理に集中する段階を、Level 3（AI-Explainable）では AI が透明性まで自ら提供し人間が事後的な監査を通じてアカウンタビリティを果たす段階をそれぞれ論じる。

各段階への移行の鍵は、Level 1 ではコンテキスト認識の設計、Level 2 ではエージェント型監督の確立、Level 3 では来歴追跡と検証可能性の確立にある。なお、Level 3 は直近の導入対象というより、技術的・社会的条件が整った先にある長期的な到達点として位置づける。アカウンタビリティという軸を据えることで、技術の進化に左右されない設計判断の基準を持つことができる。AI の能力が拡大しても、ステークホルダーに対して説明し正当化する責任は人間に残る。本稿が提示する枠組みは、この不変の原則に基づいて開発プロセスを適応させていくための指針となる。

# 背景

本稿の主張は、AI-Native 開発のプロセスはアカウンタビリティの要求水準から逆算して設計すべきだというものである。本章では、この主張の背景にあるコーディングエージェントの台頭と商用システム開発における課題を整理し、AI-Native 開発の概念を定義する。

## コーディングエージェントと Vibe Coding

コーディングエージェントの登場により、ソフトウェア開発の自動化は新たな段階に入った。GitHub Copilot、Cursor、Claude Code、Devin などのツールは、開発者の指示に基づいてコードの生成、修正、テスト、デバッグを自律的に実行する。従来のコード補完とは異なり、複数ファイルにまたがる変更や外部ツールとの連携を含む複雑なタスクを一貫して遂行できる点が特徴である。

Vibe Coding は、この技術的進歩を背景に生まれた開発スタイルである。Andrej Karpathy が2025 年 2 月に提唱したこのアプローチでは、開発者は「コードの存在を忘れて雰囲気（vibe）に身を委ねる」とし、LLM に大部分のコード変更を委ね、エラーが出れば AI に修正させるという開発スタイルを描いた [6]。探索的な開発やコンセプト検証において、従来では考えられなかったスピードでのイテレーションを可能にしている。実際、Y Combinator CEO の Garry Tan は、2025 年冬バッチの参加スタートアップの 25% がコードベースの 95% を AI で生成していたと報告しており [7]、この流れが一過性ではないことを示唆している。

## 商用システム開発への課題

しかし、Vibe Coding には限界がある。プロトタイピングやスタートアップの初期プロダクトには適しているものの、商用システムの開発にそのまま適用することは難しい。たとえば、AI が生成したコードに起因する障害が本番環境で発生した場合、生成過程が記録されていなければ原因の特定は困難になる。AI が選択した設計判断の根拠が不明なまま、保守担当者がコードを修正しなければならない状況も生じうる。さらに、顧客や規制当局から「なぜこのような実装になったのか」と問われたとき、「AI が生成した」という回答ではアカウンタビリティを果たすことはできない。

品質保証、保守性、そして何よりアカウンタビリティの観点が欠落しているのである。Roychoudhury らは、AI ソフトウェアエンジニアの実用化には従来の人間主導の開発と同等以上の信頼（trust）を確立する必要があるとし、ソフトウェア工学の焦点が「規模の拡大（programming at scale）」から「信頼の確立（programming with trust）」へ移行する可能性を論じている [4]。企業が責任を持ってソフトウェアを提供するためには、AI の能力を活用しながらも、品質とアカウンタビリティを確保するための開発プロセスを意識的にデザインする必要がある。

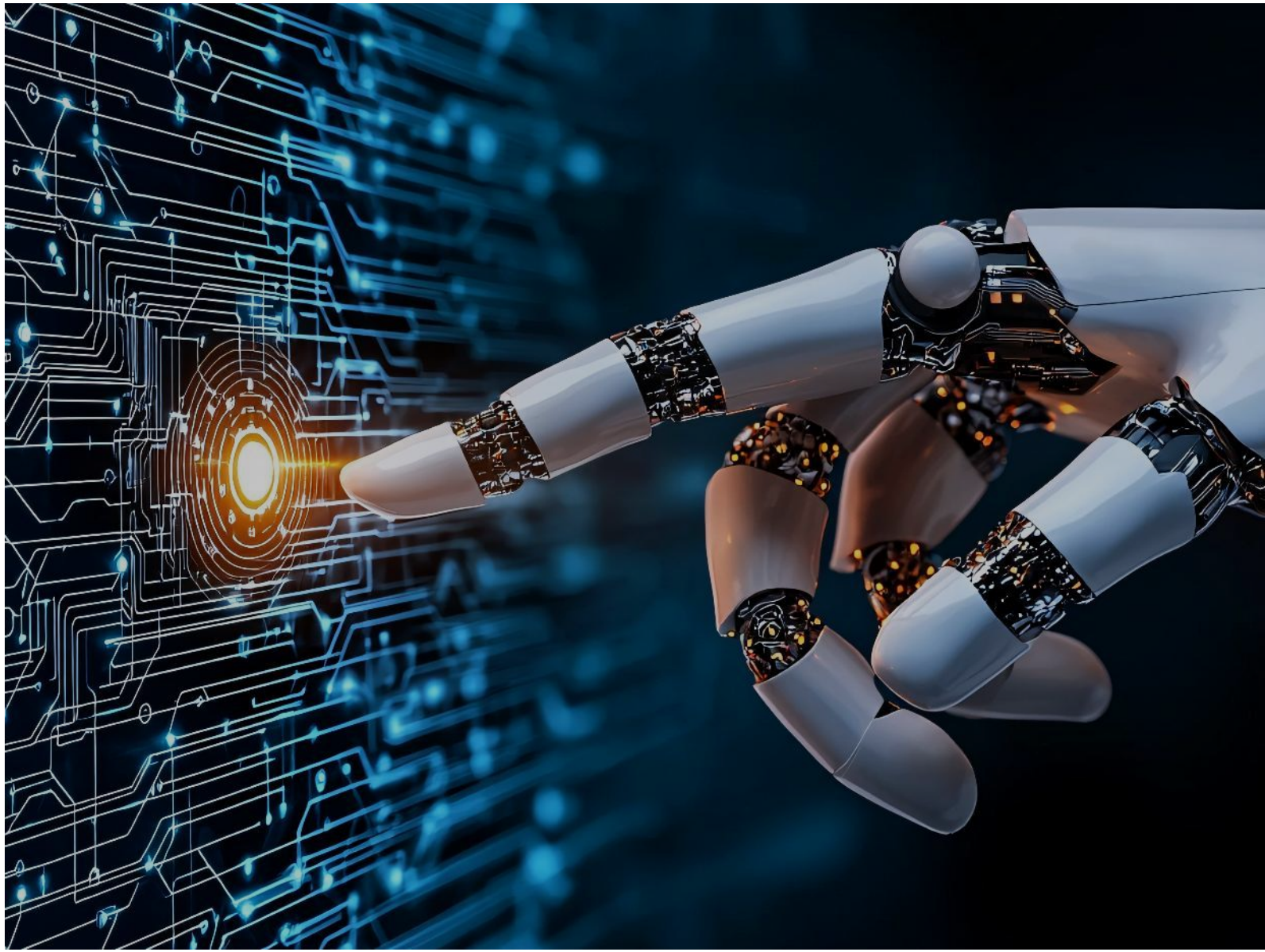
## AI-Native 開発とは

コーディングエージェントを開発の主体に据える開発スタイルは、業界全体で急速に広まりつつある。Xebia は「2026 年はソフトウェアエンジニアリングが AI-Native になる年」と予測し [8]、各企業が AI を実験的な補助ツールとしてではなく開発プロセスの中核に組み込む動きが加速している。NTT DATA でも 2025 年からこの潮流にいち早く取り組んでおり、本稿ではこうした開発スタイルを AI-Native 開発と呼ぶ。本稿で述べる知見の多くは、その実践から得られたものである。

AI-Native 開発とは、AI を開発の主体に据え、AI が最大限の能力を発揮できるよう開発環境そのものを最適化するアプローチである。「ネイティブ」とは、主体と環境の間に翻訳を必要としない状態を指す。AI を補助ツールとして既存の人間向け環境に組み込むのではなく、AI が開発環境のリソースを直接処理できる状態を構築することが AI-Native 開発の核心である。

AI-Native 開発の要点は、次の三点に要約できる。

- 開発の主体を人間中心（AIは補助利用）から、AI エージェント中心の協働へ移すこと
- AI が直接扱えるよう、要件・設計・手順・ルールをテキストと API/CLI 中心に再構成すること
- 人間の役割を実装そのものから、方向づけ・監督・説明責任へ移すこと



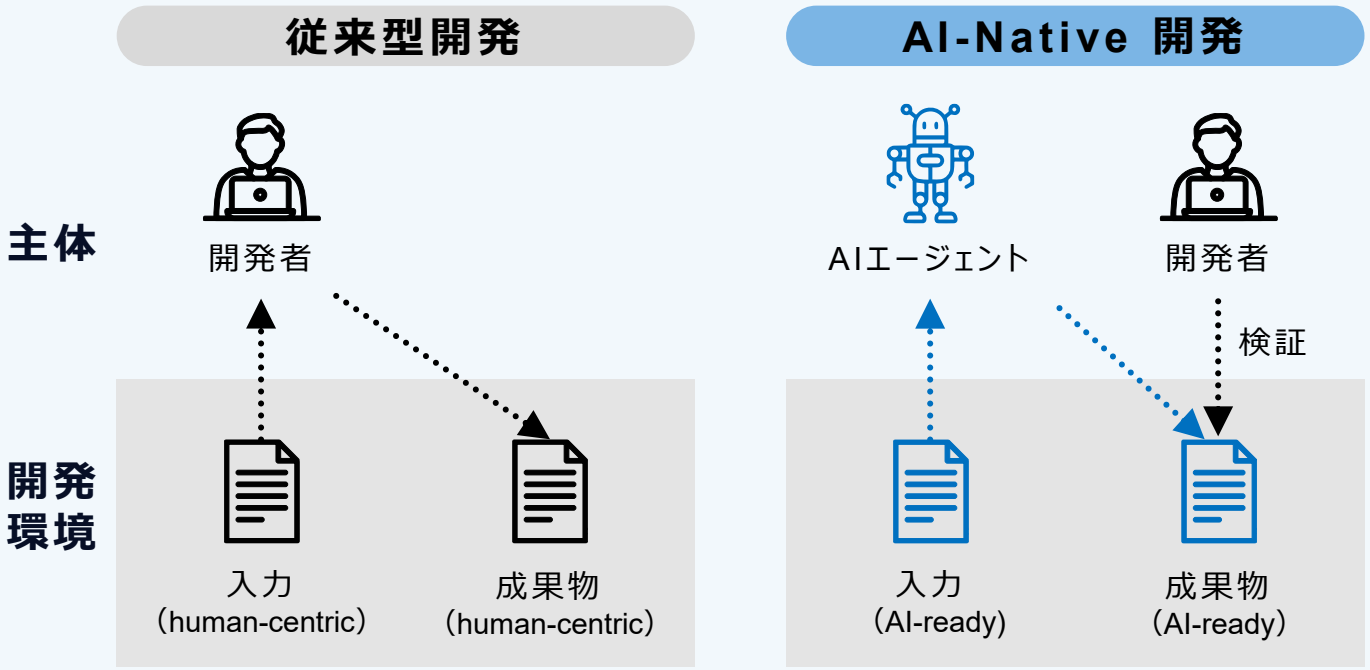



**図1.従来型環境と AI-Native 環境の比較**

図1にこの変化を示す。従来の開発環境は人間中心（human-centric）に設計されており、GUIベースのツール、MS Excel や MS Word などのバイナリ形式のドキュメント、暗黙知に依存したプロセスが前提であった。AI エージェントはこれらを直接処理できず、翻訳や変換が必要となる。AI-Native 開発では、これらを AI が処理可能な構成（AI-ready）に刷新する。具体的には、要件・設計・手順・ルールを Markdown 等のテキスト形式で管理し、暗黙知を明示的なルールとして記述し、ツールを CLI や APIベースに統一する。開発の主体が AI エージェントに移り、人間の役割は成果物の検証へと変わる。この環境の転換により、人間の作業速度に制約されていた開発プロセスが、AI の処理速度でのイテレーションへと変わる。

以降では、アカウンタビリティを軸に据えたソフトウェア開発の基本構造を整理し、AI-Native 開発のデザイン戦略について論じる。

# AI 時代の開発の基本構造

AI-Native 開発は、AI に開発を任せればそれで完了するという単純なものではない。商用システムの開発においては、動くものを作るだけでなく、品質を保証し、開発プロセスの透明性を確保し、ステークホルダーへのアカウンタビリティを果たす必要がある。AI がこれらの責務をどこまで担えるかは技術の成熟度によって異なり、そこに段階が生まれる。

本章では、この段階を理解するための基盤を三つのステップで整理する。まず、商用システム開発に求められるガバナンス構造を、IIA（The Institute of Internal Auditors）のスリーラインモデル [5] をソフトウェア開発に適用することで示す。次に、このガバナンス構造の中で果たすべき4つの責務——機能性、品質、透明性、アカウンタビリティ——を定義する。最後に、AI 時代においてこれらの責務を AI と人間がどのように分担するかを論じ、成熟度モデルを導入する。

## ガバナンス構造: スリーラインモデルの適用

スリーラインモデルは、組織のガバナンスとリスク管理の役割を三つのラインで整理するフレームワークである。このモデルをソフトウェア開発に適用することで、執行・監督・監査の責務の所在が分離され、AI の役割が拡大しても破綻しないガバナンスの整理軸が得られる。本稿では、商用システム開発に求められる責務がどのような役割構造の中で実現されるかを図2に示す。

本稿でスリーラインモデルを採用する理由は、AI-Native 開発において求められるのが単なる防御統制ではなく、価値創出の速度を維持しながらガバナンスを組み込む設計だからである。従来の Three Lines of Defense という呼称が防御寄りの印象を与えるのに対し、現在の Three Lines Model は、価値創出とリスク管理を両立させるための役割整理として理解できる。ソフトウェア開発、とりわけ AI-Native 開発では time-to-market を維持したまま説明責任を組み込む必要があるため、この整理軸との相性がよい。

図2は、左から右へ第一ライン（執行）、第二ライン（監督）、第三ライン（監査）の順に責務が並び、最上位にガバナンス体が位置する構造を示している。第一ラインは開発実務の執行とリスクの直接的なコントロールを担う。第二ラインは第一ラインから独立した立場で開発プロセスの遵守を監督する。これら二つのラインはマネジメントの傘下にあり、開発プロセスの透明性を確保する。第三ラインはマネジメントから独立した立場で統制プロセスの有効性を検証し、ガバナンス体に保証と助言を提供する。ガバナンス体はステークホルダーに対するアカウンタビリティを果たす。

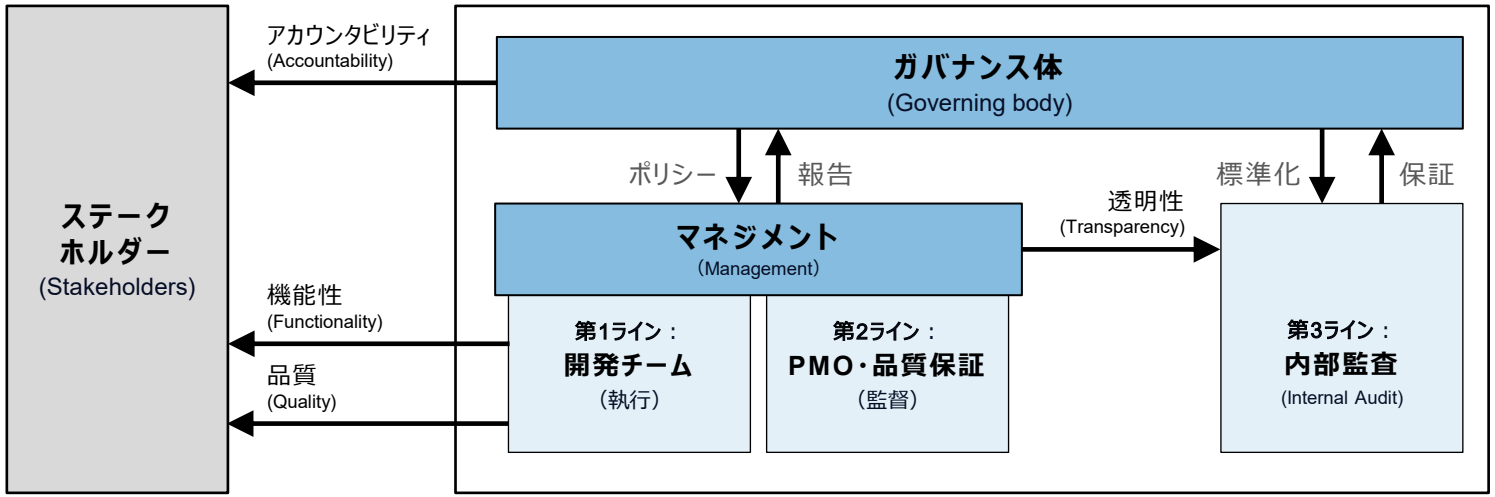



**図2.ソフトウェア開発におけるスリーラインモデル**

## 商用システムに求められる4つの責務

このガバナンス構造の中で、商用システム開発が果たすべき責務は、機能性（Functionality: 動くこと）、品質（Quality: 期待通りに動くこと）、透明性（Transparency: 開発プロセスが透明であること）、アカウンタビリティ（Accountability: 説明・正当化できること）の4つに整理できる。これら四つは、システムそのものに対する要求である機能性・品質と、開発プロセスに対する要求である透明性・アカウンタビリティを併せて捉えたものであり、同時に第一ラインからガバナンス体までの役割分担と自然に対応づけられる。以下では各責務を定義し、スリーラインモデルにおける担い手を明確にする。

## 機能性: 動くこと

機能性とは、要件で定義された機能をシステムが提供することである。スリーラインモデルにおいて、この責務は第一ラインが担う。構築、デプロイ、運用が可能な状態にあり、意図された基本機能を実行し、実際の利用を妨げることなく稼働し続けることを意味する。システムがそもそも動作しなければ、他のすべては意味を持たないため、ソフトウェアが価値を提供するための最も基本的な条件である。機能性の判定は、定義された機能が実装され動作するかどうかによって行われる。

機能性の実現において、AI は既に大きな成果を上げている。Vibe Coding [6] に代表されるスピード重視の開発スタイルでは、AI が「動くもの」を高速に生成し、プロトタイピングやコンセプト検証を劇的に加速させている。アイデアを素早く形にし、動作を確認しながら方向性を探る。このような探索的な開発において、AI の貢献は非常に大きい。また、個人やチーム内で利用するソフトウェアは、必ずしも商用システムのような高い品質やアカウンタビリティを求められないため、Vibe Coding のようなスピード重視の開発スタイルは有効である。

## 品質: 期待通りに動くこと

品質とは、システムが期待通りに動作することである。要件、仕様、ユーザーの期待に沿って振る舞い、その出力が定義された条件の下で正確、一貫、かつ信頼できることを意味する。機能性が「定義された機能を提供すること」であるのに対し、品質は「その機能が品質特性を満たした状態で動作すること」である。具体的な観点としては ISO/IEC 25010:2023 [9] のプロダクト品質モデルが定義する9つの品質特性（機能適合性、性能効率性、互換性、操作性、信頼性、セキュリティ、保守性、柔軟性、安全性）などがある。なお、2023 年の改訂で従来の 8 特性から再編され、柔軟性と安全性が追加されている。品質の判定は、定義された品質基準やテストケースに対する充足度によって行われる。

スリーラインモデルにおいて、品質は第一ラインと第二ラインの協働によって実現される。第一ラインは、リスクの所有者としてプロセス内での品質の作り込み（曖昧な要件の排除、コーディング規約の遵守、単体・結合テストの実施など）を行う。第二ラインは、第一ラインから独立した立場で、開発標準・品質基準の策定と周知、プロジェクト状況の定期的なモニタリング、品質ゲートによる工程移行判定を通じて、品質が確保されているかを監督する。加えて、セキュリティ管理部門や法務・コンプライアンス部門による専門的な観点からの検証も監督の一部として機能する。現時点では、これらの活動は主に人間の役割であるが、AI 技術の進化により、この領域も徐々に AI が担えるようになりつつある。

品質の実現には、明確な要件の定義が不可欠である。要件が曖昧であったり矛盾していたりすると、「期待通り」の基準そのものが不明確となり、品質の作り込みも検証も困難になる。これは開発の主体が人間であっても AI であっても変わらない一般的な原則である。加えて、AI によるテスト生成・実行やコードレビューの技術は急速に成熟しつつあり、品質保証の一部を AI が担う形が現実的になってきている。

## 透明性: 開発プロセスが透明であること

透明性とは、開発プロセスの実態が証跡として記録され、独立した第三者が検証可能な状態にあることである。スリーラインモデルにおいて、マネジメントは第一・第二ラインを統括し透明性を確保する責務を担い、第三ラインはマネジメントから独立した立場でこの透明性を通じて開発の実態を検証する。透明性はアカウンタビリティを実現するための前提条件でもある。アカウンタビリティが「説明できること」であるならば、透明性は「説明するための根拠が存在すること」である。

従来の開発では、人間がプロセスに沿って作業し、その結果を証跡として残すことで透明性を確保してきた。一方、AI が開発の主体となる AI-Native 開発では、AI による作業プロセスがブラックボックス化しやすいという新たな課題が生じる。具体的には、少なくとも以下の情報が記録・管理されていることが透明性の最低条件となる。

- 意思決定ログ — 設計判断や方針選択の理由と経緯
- 実行ログ — AI エージェントの作業履歴と生成過程
- 入力コンテキストの版管理 — AI に与えた指示・参照情報とそのバージョン
- 承認記録 — 人間によるレビュー・承認の結果と判断根拠

例えば、認証機能の追加を AI エージェントに依頼する場面では、「既存の認可方式に合わせてOAuth2 を採用した理由」を意思決定ログに記録し、「どのプロンプトでどのファイルを変更し、どのテストを実行したか」を実行ログに残し、「参照した設計書とセキュリティ標準の版」を入力コンテキストとして紐づけ、人間が最終承認した判断理由を承認記録として残す。この四点が揃ってはじめて、後から第三者が変更の妥当性を検証できる。

## アカウンタビリティ: 説明・正当化できること

商用ソフトウェア開発において、アカウンタビリティは欠かすことのできない要素である。システムがなぜ特定の出力を生成し、なぜそのように振る舞うのかを説明できなければ、企業はそのソフトウェアを責任を持って提供することができない。問題が発生した際に誰がどのように対処するのか、意思決定を追跡・レビュー・弁護できることが求められる。

このアカウンタビリティが人間に帰属し続ける根拠は、制度的・構造的な必然にある。法的には、契約の当事者や規制の対象となるのは法人を含む法的主体であり、AI はその主体となりえない。組織的には、監査報告や品質保証における最終的な署名者として責任を引き受ける人間が不可欠である。加えて、アカウンタビリティはステークホルダーとの利害関係の中で成立する概念であるが、AI は利害関係を持たず、責任を負う動機も能力も持たない。Ulloa らが「アカウンタビリティは人間以外の主体に委任できない」と指摘する通り [2]、AI の役割がいかに拡大してもアカウンタビリティの帰属先は人間に残る。

このモデルが示す重要な点は、アカウンタビリティが単独の活動ではなく、組織構造全体を通じて実現されることである。第一ライン（執行）が動くものを作り、第二ライン（監督）がその品質を監督し、マネジメントがプロセスの透明性を確保し、第三ライン（監査）が独立した視点で検証し、ガバナンス体がステークホルダーへのアカウンタビリティを担う。この多層的な構造があってはじめて、商用システムのアカウンタビリティは成立する。

## AI 時代の変化: AI が担う範囲の拡大

従来のソフトウェア開発では、機能性、品質、透明性、アカウンタビリティのすべてを人間が担ってきた。人間が設計し、人間がコードを書き、人間がテストを実行し、人間がレビューを実施する。さらに人間がマネジメントを通してプロセスの透明性を確保し、ステークホルダーに対するアカウンタビリティを果たす。この構造は長年にわたって機能してきたが、同時に人間の能力がボトルネックとなる構造でもあった。開発速度は人間の作業速度に制約され、品質は人間の注意力と専門性に依存し、スケーラビリティには限界があった。

AI 時代においては、むやみに AI を導入するのではなく、前節までで整理した役割構造の中で AIと人間がどのように責務を分担するかを設計し、適切な形で AI を導入することが重要である。AI 技術の進化に伴い、AI が担える範囲はこのモデルの中で段階的に拡大していく。本稿では、この拡大の過程をベースラインを含む3つの成熟度レベルとして定義する（図3）。

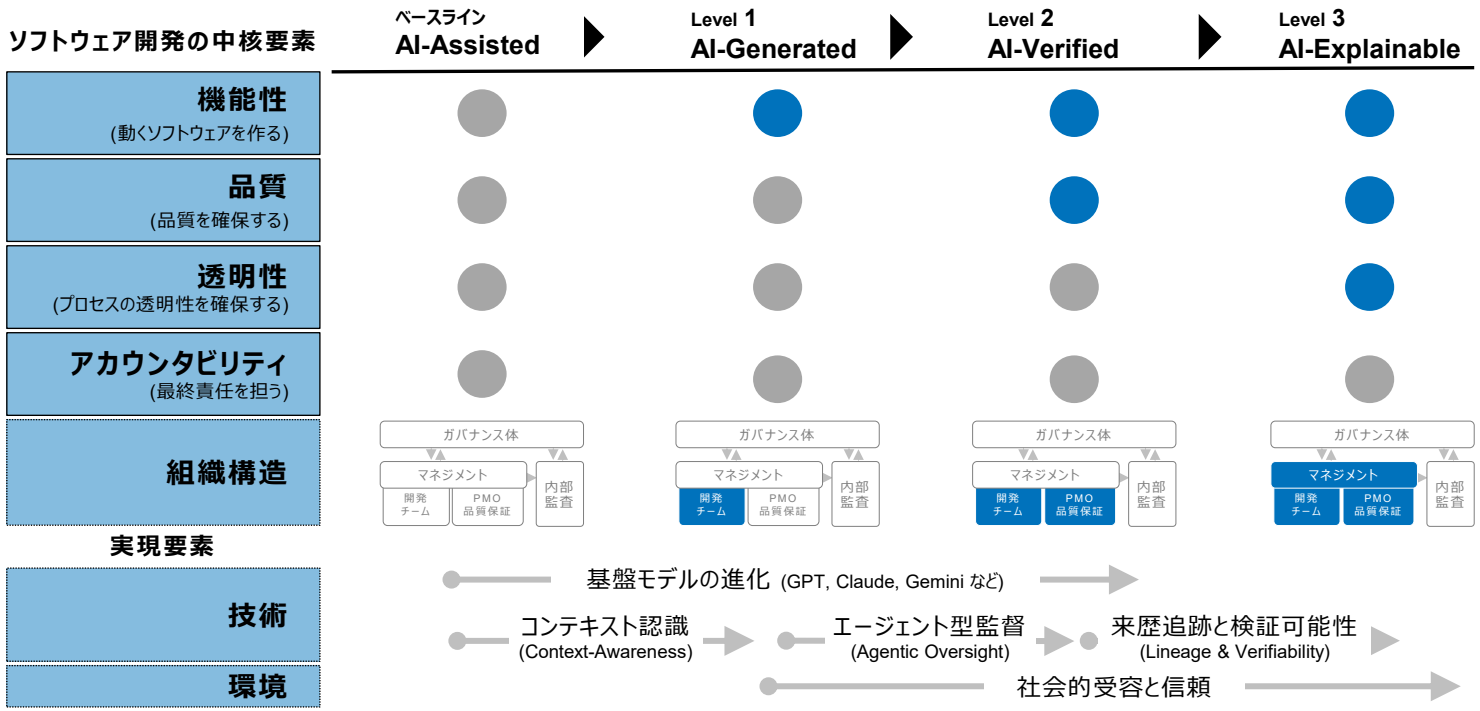



**図3. AI-Native ソフトウェア開発成熟度レベル**

図3は、左から右へレベルが上がるにつれ、AI が担う責務の範囲（機能性→品質→透明性）が拡大し、人間の役割がより上位の監督・アカウンタビリティに集約される過程を示している。各レベルの名称は、その段階で AI が新たに担う役割の特性を表している。

- **ベースライン（AI-Assisted）：**
  AI が補助ツールとして機能する現在の主流。機能性・品質・透明性・アカウンタビリティのすべてを人間が担う。多くの組織は現在この段階に位置しており、AI をコード補完などの補助ツールとして活用している。
- **Level 1（AI-Generated）：**
  AI が開発実務を担う段階。コンテキスト認識（ContextAwareness）の設計が移行の鍵となる。
- **Level 2（AI-Verified）：**
  AI が品質の監督も担う段階。エージェント型監督（AgenticOversight）の確立が移行の鍵となる。
- **Level 3（AI-Explainable）：**
  AI が開発プロセスの透明性まで自ら提供する段階。来歴追跡と検証可能性（Lineage & Verifiability）の確立が移行の鍵となる。

レベルが上がるにつれて AI が担う範囲が拡大し、人間の役割はより上位の監督・意思決定へと集約されていく。各レベルの具体的なデザインは後章で詳述する。この成熟度モデルのポイントは商用ソフトウェアの開発において、今後 AI が担う範囲が拡大しても第三ライン（監査）とガバナンス体の役割は人間に残るということである。これらの役割は、アカウンタビリティを果たすための重要な要素であり、AI が担うことが難しい領域である。

各レベルにおける AI と人間の役割分担を、実務の観点から以下の表に整理する。

各レベルへ移行するために必要な能力も異なる。Level 1 への移行では、AI が扱える環境コンテキストと指示コンテキストを整備し、レビュー可能な粒度でタスクを分解できることが前提となる。Level 2 への移行では、品質基準の形式知化、テスト自動化、監督エージェントを運用できる監視基盤が必要となる。Level 3 への移行では、来歴追跡、検証可能性、自己説明の信頼性を担保する技術基盤に加えて、それを受容する組織的・社会的条件が求められる。

| 観点 | Level 1:<br>AI-Generated | Level 2:<br>AI-Verified | Level 3:<br>AI-Explainable |
|---|---|---|---|
| **入力（確定主体と要求精度）** | 要件・ハイレベル設計を<br>人間が確定 | 要件を人間が高精度で定義し、<br>AI が要件品質を検証 | 要件を人間が定義し、<br>AI が解釈・補完 |
| **実行（AI と人間の分担）** | AI が成果物を生成、<br>人間がレビュー | AI が執行・監督を自律的に遂行 | AI が執行・監督・プロセス管理を<br>自律的に遂行 |
| **検証（品質担保の手段）** | 人間によるレビューと自動テスト | 監督エージェントによる品質ゲート判定 | AI の自己説明と<br>人間による独立監査 |
| **証跡（記録する成果物）** | レビュー記録、テスト結果 | 品質ダッシュボード、<br>エスカレーション記録 | 来歴追跡記録、監査証跡 |
| **エスカレーション（人間介入の条件）** | 成果物のレビュー時 | 重大な意思決定ポイント、<br>品質基準逸脱時 | 監査で不整合を検出した時 |

# デザインのフレームワーク

前章では、商用システム開発に求められるガバナンス構造と、AIが担う範囲が段階的に拡大していく成熟度モデルを整理した。しかし、各段階への移行は技術の進化だけで自動的に実現するものではない。技術的な成熟に加えて、ステークホルダーの信頼と社会的な受容が伴って初めて移行は実現する。こうした段階的な移行を意識的にデザインするためのフレームワークとして、本章ではPPT（Process, People, Technology）の三要素に基づくアプローチを提示し、各観点に沿ったデザインの指針を示す。
本章の要点は、AI-Native　開発の成否が個別ツールの性能だけでは決まらず、Process、People、Technology を一体として設計できるかにかかっているという点にある。以降では、まず PPT という共通の見取り図を示し、その上で各観点における設計の勘所を整理する。

## PPT フレームワーク

PPT フレームワーク（Process, People, Technology）は、技術導入のデザインにおいて広く用いられてきた考え方である。新たな技術を組織に導入する際には、技術そのものだけでなく、それを活用するプロセスと人の要素を併せて設計する必要がある。

- **Process（プロセス）：**
  技術を組み込んだ業務プロセスの設計
- **People（人）：**
  技術を活用する人材の育成と役割定義
- **Technology（技術）：**
  適切なツールやインフラの選定と整備

この三要素は相互に依存しており、いずれか一つだけを最適化しても効果は限定的である。例えば、ルールベースの自動化の時代においても、CI/CD　パイプラインやテスト自動化ツール

（Technology）を導入するだけでは不十分であった。組織が自動化の恩恵を得られたのは、自動化を前提としたビルド・デプロイのプロセス（Process）を設計し、自動化ツールを運用・改善できるエンジニア（People）を育成したときである。PPT フレームワークは、このように技術導入を三要素の統合的なデザインとして捉える視点を提供する。

AI-Native 開発においても、デザインの基本構造は PPT に従う。ただし、AI-Native 開発では Technology の内部構造がより重要になる。従来の自動化ツールはルールベースであり、設定やスクリプトで動作が決定されていた。

一方、AI エージェントは自然言語の指示とコンテキストを解釈し、自律的に判断して動作する。そのため、適切な指示（Instruction）と、AI が処理可能な形式で整備された環境（Environment）の設計が成否を左右する。すなわち、Technology の中に AI エージェントという新たな主体が加わり、その主体に対してさらに指示と環境を設計するという、PPT が入れ子になる構造が生まれる（図4）。

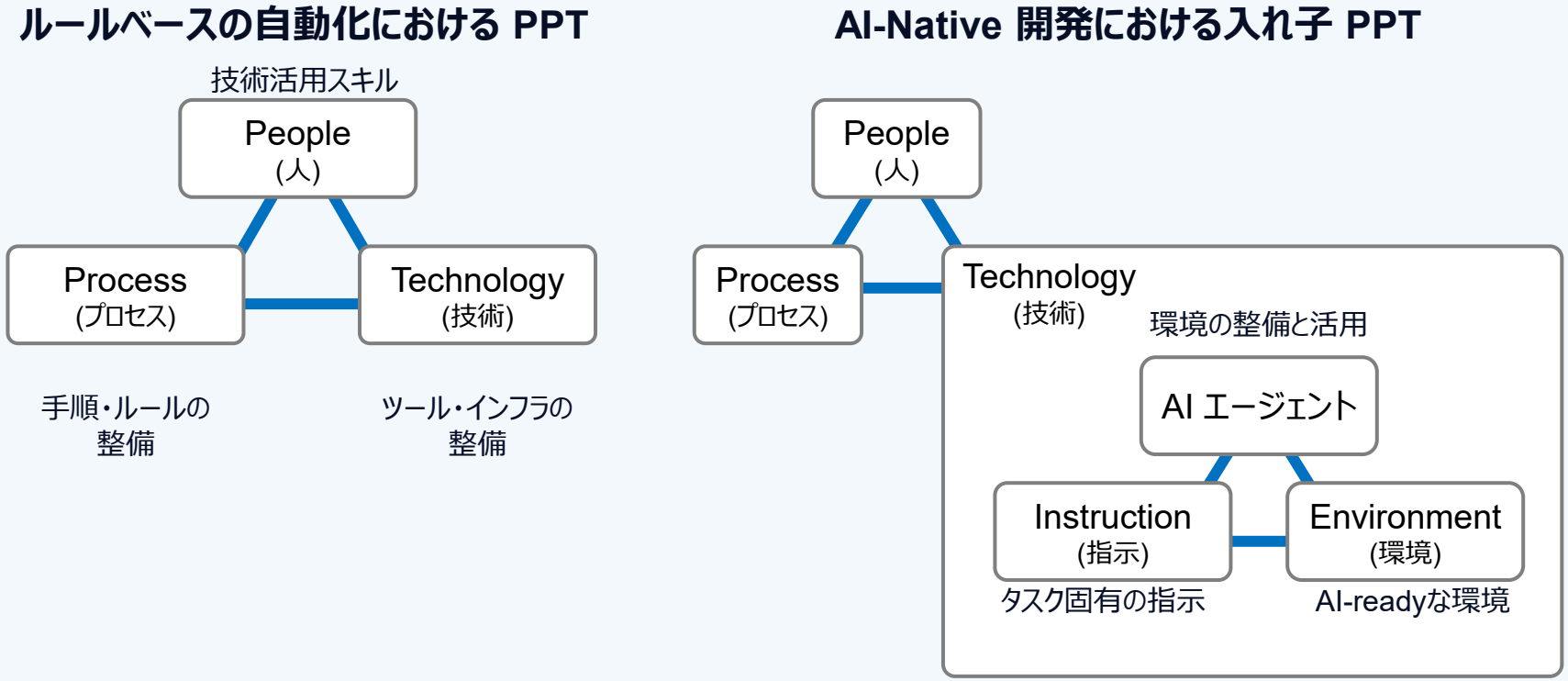



**図4.自動化の時代と AI-Native 時代における PPT フレームワークの入れ子構造**

## デザインの指針

以降では、PPT の各観点に基づく AI-Native 開発のデザインの指針を示す。Process ではアカウンタビリティを軸にしたプロセス設計、People では暗黙知の形式知化と役割の移行、Technologyではコンテキストの設計を論じる。

## Process: アカウンタビリティを軸にしたプロセス設計

いかに効率的な開発が実現できても、アカウンタビリティを果たせなければ商用システムとして提供することはできない [2]。したがって、プロセスの設計はプロジェクトが求めるアカウンタビリティの水準を起点とすべきである。前章で述べた通り、透明性はアカウンタビリティを実現するための前提条件である。求められるアカウンタビリティの水準が開発プロセスに求める透明性のレベルを決定し、透明性の要求が AI エージェントに与える自律性の度合いを規定する。AI エージェントの自律性とプロセスの透明性の間にはトレードオフの構造がある。

AI エージェントの挙動を固定するほど透明性の確保コストは下がるが、AI の柔軟性が制約され生産性が低下する。逆に、自 律性を高めるほど生産性は向上するが、履歴の保存・分析の仕組みが必要となり監査コストが増大する。プロジェクトが求めるアカウンタビリティの水準に応じて、このトレードオフの中で適切 なバランス点を選択することがプロセス設計の核心である（図5）。

図5は、横軸に AI エージェントの自律性の度合い、縦軸に透明性の確保コストをとり、アカウンタビリティの要求水準がこのトレードオフの中でバランス点を規定する関係を示している。この連続的な段階における三つのアプローチの特徴を以下に整理する [3]。

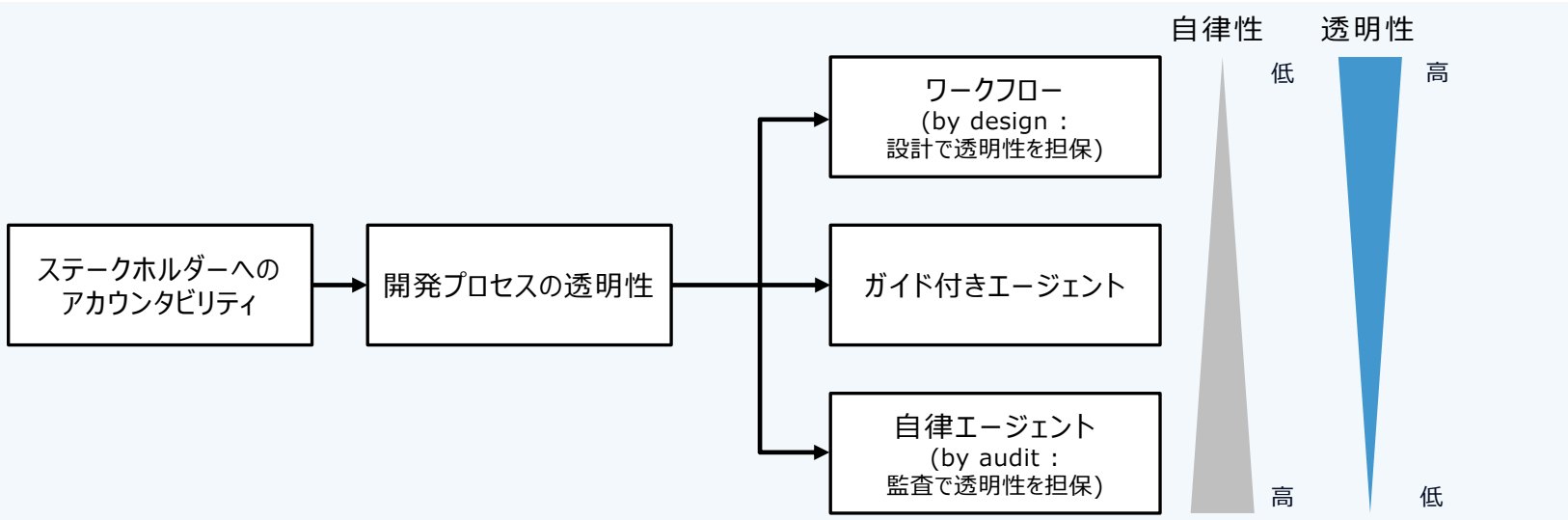



**図5.アカウンタビリティを起点としたプロセス設計 — 透明性と自律性のトレードオフ**

### ■ ワークフロー（Workflow）：

AI エージェントの動作をあらかじめ定義された手順に沿って実行させる。手順を定義した時点で透明性が事前に保証される（by design）ため、再現性と検証可能性も自然に担保され、アカウンタビリティの要求が高い工程に適している。

### ■ ガイド付きエージェント（Guided Agent）：

遵守すべきルールや判断基準をコンテキストとして与えたうえで、具体的な作業手順はエージェントに委ねる。ワークフローより柔軟性が高い一方、ルールが実際に遵守されたかどうかを確認する仕組みが必要となる。

### ■ 自律エージェント（Autonomous Agent）：

目標と制約条件を与え、判断の大部分をエー ジェントに委ねる。最も柔軟性が高い反面、エージェントがどのような手順を踏んだかは実行してみるまでわからない。透明性は事前に保証されず、作業の履歴を保存し事後に検証する（by audit）必要がある。

一つのプロジェクト内であっても、タスクの性質に応じてこれらのアプローチを使い分けることが現実的である。例えば、セキュリティに関わる認証処理の実装は証跡の確保が求められるためワークフローとして手順を固定する一方、UI コンポーネントの実装は自律エージェントに委ねるといった使い分けが考えられる。

なお、この段階における重心は成熟度レベルによっても変化する。Level 1 では人間のレビューにより品質を担保するため AI エージェントの自律性を制約することに優位性がある。一方で成熟度が上がり AI が監督や透明性の確保を担えるようになるにつれ、より自律的なアプローチを採用できる余地が広がる。

このプロセス設計において、チェックポイントは品質ゲートであると同時に、人間が持つコンテキストを開発プロセスに注入する機会でもある。AI エージェントはセッションごとにステートレスであり、人間が暗黙的に持つ設計意図やドメイン制約を自ら獲得することができない。レビューやチェックポイントの本質的な機能は、こうした人間のコンテキストを AI の成果物に反映させることにある。この視点は、次節で述べるコンテキストエンジニアリングと暗黙知の形式知化を、プロセス設計の中で統合する接点となる。チェックポイントの頻度と配置の設計は、すなわちコンテキスト注入のタイミングと量の設計であり、この関係は成熟度レベルの進展に伴って変化する。

## People: 暗黙知の形式知化・AI 監督スキルと役割の移行

AI-Native 開発において、人の役割は根本的に変化する。従来は開発者自身がコードを書き、テストを実行し、品質を作り込んでいた。AI-Native 開発では、人間の役割は「開発の実行者」から「AI の方向づけと検証を行う監督者」へと移行する。

この役割の移行において特に重要なのは、暗黙知の形式知化である。AI エージェントはセッションごとに状態がリセットされ、経験や暗黙知を蓄積することができない。従来のチーム開発では、「なぜこのアーキテクチャを選んだのか」「過去にどのような問題が起きたのか」といった知識がチームメンバーの頭の中に暗黙的に蓄積され、日常のコミュニケーションを通じて共有されてきた。AI-Native 開発では、こうした暗黙知をテキストとして明示的に記述し、AI エージェントが参照できる形で整備する責務が人に求められる。

もう一つ重要なスキルは、AI の出力を独立した立場から批判的に評価する能力である。AI エージェントが生成するコードや設計判断は一見もっともらしく見えるため、人間はその正しさを過信しやすい。こうした傾向は自動化バイアス（Automation Bias）として航空や医療の分野で広く知られており、自動化システムの出力を無批判に受け入れることで重大な見落としが生じるリスクが指摘されてきた。AI-Native 開発でも同様のリスクが存在し、AI の出力に対して「なぜこの設計なのか」「見落としている制約はないか」と問い続ける姿勢が求められる。

しかし、このスキルは容易には習得できない。AI の出力を評価するには対象ドメインへの深い理解が前提となるうえ、もっともらしい出力に対してあえて疑問を持つという行為は認知的な負荷が高いためである。組織としてレビューの観点を明文化し、訓練の機会を設けるといった意図的な取り組みが必要となる。

人の役割は成熟度レベルに応じてさらに変化する。Level 1 では AI の出力を直接レビューする品質の番人、Level 2 では AI による品質監督プロセスの管理者、Level 3 では AI の判断に対するアカウンタビリティを担う監査者へと移行する。各レベルにおける具体的な役割と求められるスキルについては、各レベルの章で詳述する。

## Technology: コンテキストの設計

前章で述べた通り、AI が補助ツールから開発の主体へと移行するための鍵はコンテキスト認識の設計にある。コンテキスト認識とは、開発タスクの前提や周囲の状況を踏まえて適切な出力や判断を行う能力を指す。この能力はモデル単体の性能で決まるものではなく、AI エージェントにどのような情報をどう与えるかという設計によって実現される。この設計、すなわちコンテキストエンジニアリングが Technology における中心課題となる。

コンテキストの設計空間は、環境コンテキストと指示コンテキストの二層に構造化できる。この二層は、図4で示した入れ子の PPT 構造における Environment と Instruction の区分に対応する。

### ■ 環境コンテキスト:

プロジェクト全体で共有される永続的な前提である。
コーディング規約、設計方針、アーキテクチャの選定理由、用語定義、既存のコードベースや設計書などが該当し、一度整備すれば比較的長期間にわたって有効であり、AI エージェントが個々のタスクに取り組む際の判断基盤として機能する。

### ■ 指示コンテキスト:

個々のタスクに固有の一時的な情報である。タスクの目的、スコープ、受入条件、関連ファイル、テスト結果、CI ログなどが該当し、タスクごとに構成されて AI エージェントの具体的な行動を方向づける。

これら二層は、取得・管理・提供の方法においても性質が異なる。環境コンテキストは主に人間が作成・維持する。規約や方針は人間が策定し、設計判断の背景は意思決定者が記録する。ただし、コードベースや設計書といった既存の開発成果物は情報量が大きく、そのままでは AI エージェントのコンテキストに収まらない。必要な情報を検索・取得する仕組みや、要約して提供する仕組みを整備することが、環境コンテキストの実効性を左右する。一方、指示コンテキストは、人間が指定する要素（タスクの目的や受入条件）と、ツールから自動収集される要素（関連するソースファイル、テスト結果、CI ログなど）を組み合わせてタスクごとに構成される。

例えば、「既存 API に監査ログ出力を追加する」というタスクでは、環境コンテキストとしてコーディング規約、既存のログ出力方針、監査項目の命名規則、対象サービスのアーキテクチャ方針を与える。一方、指示コンテキストとしては、今回変更するエンドポイント、追加すべきログ項目、受入条件、関連するテスト失敗ログを与える。このように二層を分けて設計することで、AI が参照すべき恒常的ルールと今回タスク固有の要求を混同しにくくなる。

この二層のコンテキストは、Technology 単体で完結するものではなく、Process と People の設計と連携することではじめて機能する（図6）。People が暗黙知を形式知化して環境コンテキストを生み出し、Process のチェックポイントが指示コンテキストの注入タイミングを規定し、 Technology がコンテキストの構造と品質を設計する。この三要素の連携によって、AI エージェントへの適切な情報提供が実現される。

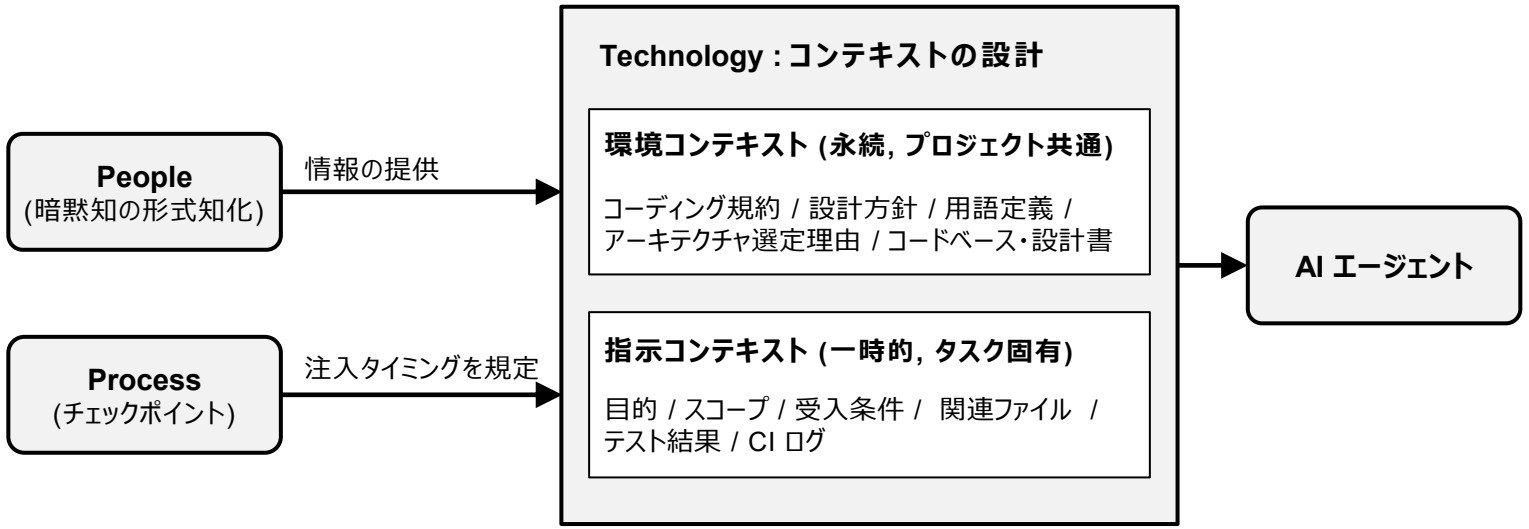



**図6.コンテキストの二層構造と PPT の連携**

コンテキストの設計においては、情報の適量性と鮮度の維持が基本原則となる。コンテキストが複雑になりすぎると AI エージェントの処理精度は低下するため、タスクを適切な粒度に分解し、各タスクに必要十分な情報を選定して与えることが重要である。また、コンテキストの鮮度も品質に直結する。設計方針の変更やアーキテクチャの見直しが環境コンテキストに反映されていなけれ ば、AI は古い前提に基づいて成果物を生成する。環境コンテキストをコードと同様にバージョン管理し、変更の追跡と整合性の維持を行うことで、コンテキストの信頼性を担保する。

これらの設計原則は全レベルに共通するが、設計の重心はレベルに応じて変化する。レベルが上がり AI の自律性が高まるにつれ、コンテキストの対象は開発タスクの方向づけから、品質の判定基準、さらにはアカウンタビリティを支える記録へと広がる。各レベルにおける具体的な設計手法は後章で詳述する。

# Level 1: AI-Generated — AIが動くものを作り、人が品質を作り込む

前章では、PPT フレームワークとデザインの指針を提示した。本章では、これらを成熟度モデルの第一段階に適用し、具体的なデザインを示す。コーディングエージェントの急速な進化により、AIが開発の主体として機能するこの段階は技術的に実現可能になりつつある。NTT DATA が 2025 年から実践を進めている領域であり、本章で述べる内容の多くはその実践から得られた知見に基づいている。この段階への移行は、AI エージェントにプロジェクト固有のコンテキスト（コーディング規約、設計方針、判断基準）を整備して与えること、開発情報をテキストベースで管理する AI-Native な環境を構築すること、そして AI の出力を監督するスキルとプロセスを確立することが前提となる。

## この段階の特徴

この段階では、AI がコードだけでなく設計書やテストコードを含む開発成果物全体を生成し、人間がレビューを通じて品質を作り込む。人間が上流工程で方針を確定させ、下流工程に向かうほど AI が成果物を生成する構造をとる。方針が明確であるほどレビューの観点も明確になり、AI の成果物を効率的に検証できるためである。この段階がもたらす生産性の変化は、開発者一人あたりのアウトプット量の飛躍的な増大である。従来は開発者が自ら行っていた設計・実装・テストを AI が担うことで、開発者の役割は「コードを書く」から「AI の出力を方向づけ、検証する」へと変わる。NTT DATA の実践においても、従来は数日を要していた機能の実装が数時間で完了するケースが確認されている。

ただし、この段階には固有の構造的限界がある。AI の生成速度は人間のレビュー速度を大幅に上回るため、開発スループットは人間のレビュー能力に制約される。この限界を前提として、レビュー効率を最大化する PPT デザインが Level 1 の核心的な課題となる。この段階の鍵となる実現要因はコンテキスト認識である。以下では、グリーンフィールド開発における PPT デザインを示した上で、ブラウンフィールド開発固有の課題を論じる。

## グリーンフィールド開発

グリーンフィールド開発は、既存の制約がない状態で新規にシステムを構築するパターンである。 AI-Native 開発に最適化された環境を一から設計できるため、PPT フレームワークを理想的な形で適用できる。図7に、Level 1 のグリーンフィールド開発における協働プロセスの全体像を示す。

図7は、上流（要件定義・ハイレベル設計）を人間が確定させ、下流（詳細設計・実装・テスト）を AI が生成し、人間がレビューで品質を作り込む一連の流れを時系列で示している。

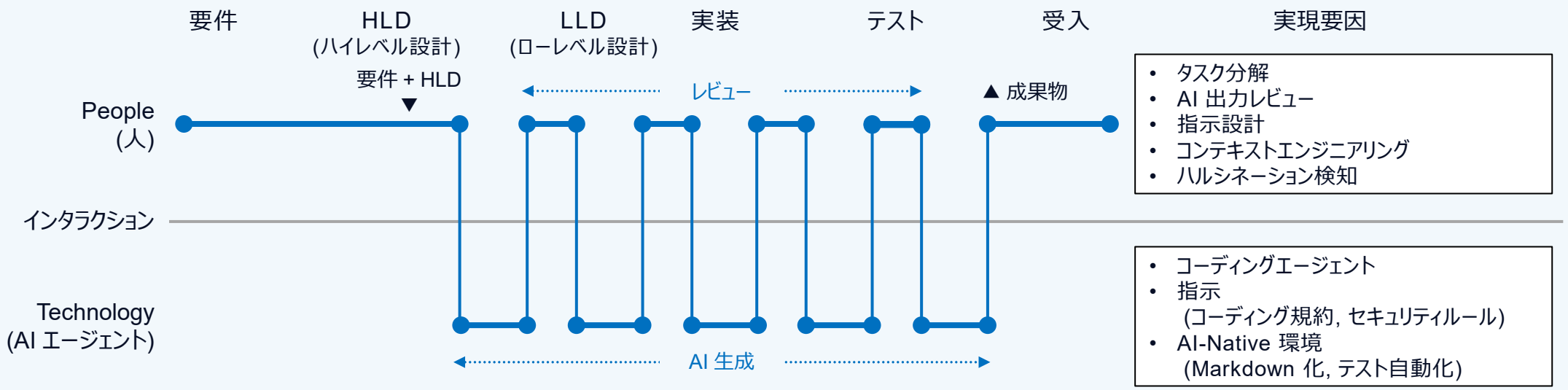



**図7. Level 1 における協働プロセスの全体像**

## Process: AI と人の協働プロセス

コーディングエージェントの基本的な活用形態は、開発者が単一のエージェントと対話しながら開発を進めるスタイルである。開発者がタスクを指示し、エージェントが成果物を生成し、開発者がレビューを通じて誤りや意図との相違を修正するサイクルを繰り返す。開発者が常にループの中にいるため、透明性の確保とアカウンタビリティの維持が容易である。

この基本的なサイクルを効果的に回すために、Level 1 のプロセス設計では以下の三つの要素を定める。人間が確定させる上流工程の範囲と粒度、AI に委ねる下流工程の自律性の度合い、そして人間によるレビューの観点と頻度である。

Level 1 では人間が要件定義とハイレベル設計を確定させ、AI が詳細設計・実装・テストを含む成果物を生成する。ハイレベル設計とは、アーキテクチャの選定、モジュール間の責務分割、インターフェース定義（API 仕様、データ型）など、モジュール間の境界と契約を指す。人間がこの境界を確定させることで、AI の成果物の影響範囲が限定され、レビューの焦点が明確になる。大きな要求はマイクロ要件（ユーザーストーリー単位）に分割し、各要件ごとにこのサイクルを回す。

AI に委ねる下流工程においては、前章で述べた透明性と自律性のトレードオフに基づき、タスクのリスク特性に応じてアプローチを使い分ける。Level 1 では人間がレビューで品質を担保するため、自律性を制約する方向に重心を置くことが基本となる。リスクの高い工程では人間によるレ ビューを必須とするチェックポイントをワークフローに組み込み、確認漏れを防ぐ。

人間によるレビューは Level 1 における品質確保の中核であり、次の観点で実施する。

第一に、要件との整合性である。AI の成果物が上流で確定した要件・ハイレベル設計に沿っているかを確認する。インターフェース定義との適合、機能要件の網羅性が対象となる。

第二に、コードの正確性と保守性である。ロジックの誤り、エッジケースの欠落、可読性の低下がないかを確認する。

第三に、非機能要件の充足である。性能やセキュリティの要件を実際に満たしているか、テストカバ レッジが十分かなど、実行・計測を伴う検証を行う。レビューの頻度は、タスクの粒度やリスクに応じて調整する。

## People: AI を監督するスキル

AI-Native 開発では、開発者の役割が「コードを書く人」から「タスクを分割し AI を監督する人」へと変化する。Process 節で述べた上流から下流への流れに沿って、求められるスキルを整理する。

第一に、チーム共通の指示と環境を設計・維持する能力が求められる。AI の出力品質は個々の開発者のプロンプト技術だけでなく、チームで共有するコーディング規約、テスト方針、AI が参照するドキュメントの品質に大きく依存する。個々の開発者の AI 活用スキルに依存するのではなく、指示と環境の品質を通じてチーム全体の生産性を底上げすることが重要である。

第二に、個々のタスクに対するコンテキストエンジニアリングの能力が求められる。タスクを AIが処理可能な粒度に分解し、各タスクに必要十分なコンテキスト（関連ファイル、設計方針、制約条件）を選定し、AI が誤解しない明確な指示を記述する能力である。コンテキストの過不足は AIの出力品質に直結するため、このスキルは生産性を左右する。

第三に、AI 出力のレビューには固有のスキルが求められる。大量のコードを短時間で評価する速読力、AI が生成しやすい典型的な誤りパターン（幻覚による存在しない API の使用、コンテキスト外の仕様の混入、一見正しく見えるが要件を微妙に逸脱したロジックなど）を見抜く目、そして AI の出力を鵜呑みにせず批判的に評価する姿勢である。

第四に、前章で述べた自動化バイアスへの対処が必要である。Level 1 では品質保証の最終的な責任は人間にあるため、レビューの実効性を維持する仕組みが不可欠である。AI の出力にはテストの合格や静的解析の通過を必須とすることで機械的に検証可能な品質基準を設け、人間のレビューは設計意図との整合性やビジネスロジックの妥当性など、自動化が困難な観点に集中させる。機械的な検証と人間の判断を分離することで、レビューの焦点が明確になり形骸化を防ぐ。

## Technology: AI の動作環境と指示の整備

Technology は環境と指示の設計から構成される。環境設計の大前提は、コーディングエージェントがテキストの読み書きに長けているという特性を活かし、あらゆる開発情報をテキストで管理することである。Level 1 ではコーディングエージェントを要件整理、設計、テスト設計を含む全工程で活用するため、すべての成果物が AI にとって処理可能なテキスト形式であることが不可欠となる。

この前提に基づき、環境の設計では AI と人間の知識ギャップを埋めることが核心となる [1]。非テキスト情報をテキスト情報に変換し、AI が直接処理できる形にする。具体的には、要件・設計・手順・ルールを Markdown で管理し、図も Mermaid 等でテキストから生成できる形式を採用する。技術スタックは LLM が学習データとして多く触れているメジャーで標準的なものを選ぶ。テストは可能な限り自動化し、AI エージェントが実行・検証のループを回せるようにする。

指示の観点では、コーディング規約、成果物の作り方、テスト手順、ソフトウェア仕様などを明文化し、AI が迷うポイントをルール化する。判断基準が一貫しないと AI の出力にもばらつきが生じるため、「誰が見ても同じ判断になる」状態を目指す。セキュリティについても、入力値のバリ デーション方針、認証・認可の実装パターン、機密データの取り扱いルールなどを指示として明文化し、AI が生成するコードのセキュリティ品質を底上げする。機密情報（API キー、認証情報など）がプロンプトに含まれないよう、シークレット管理も徹底する。

## ブラウンフィールド開発

ブラウンフィールド開発は、既存システムの改修・保守を行うパターンである。企業における開発の多くはブラウンフィールド開発であるため、AI-Native 開発の効果を最大化するためにはこの領域への適用が不可欠である。

ブラウンフィールド開発には、既存資産の活用範囲に応じていくつかのパターンがある。保守・機能追加は、既存システムの機能を維持しながら改修や機能追加を行う。リホストは、アプリケーションの機能や設計を変更せず、実行環境のみを移行する。リライトは、既存システムの振る舞いと構造を理解した上で、実装を全面的に書き直す。リビルドは、既存システムの要件を起点に、設計から実装まで全面的に再構築する。

## Process: 段階的な移行と既存制約への適応

グリーンフィールド開発と基本原則は共通するものの、既存システムへの影響範囲分析や回帰テスト設計などがプロセスに加わる。また、既存システムの技術的制約（レガシーフレームワーク、独自ライブラリ等）により AI に委ねられる範囲が限定される場合があり、どの工程・モジュールで AI を活用し、どこを人間が担うかの判断をプロジェクト固有の制約に応じて行う必要がある。

保守・機能追加パターンでは、過去の開発資産が膨大であり、一括で AI-Native 開発向けに整備することは現実的ではない。そこで、段階的な移行のアプローチが有効である。既存の開発プロセス全体を一度に置き換えるのではなく、新規の変更要求を AI-Native な形で対応し、対象領域を徐々に拡大していく。このアプローチはモノリシックシステムのマイクロサービス化で知られる Strangler Fig パターンと類似している。どの領域から着手するかについては、変更頻度が高い領域（投資対効果が高い）、ドキュメントが比較的整っている領域（整備コストが小さい）、利用価値の高い情報を持つ領域を優先する。段階的移行の過程では AI 向けの開発環境と従来の開発環境が併存するため、どの領域が移行済みかをチーム全体で管理・共有する仕組みが重要となる。

## People: 有識者による暗黙知の形式知化

グリーンフィールド開発で述べたスキル（組織的役割定義、コンテキストエンジニアリング、AI 出力レビュー、自動化バイアス対処）はブラウンフィールド開発でも引き続き求められる。加えて、既存システムのドメイン知識を持つ有識者の役割が決定的に重要となる。グリーンフィールド開発では指示と環境を新規に設計すればよいが、ブラウンフィールド開発では既存システムの設計判断の背景、過去の障害対応の知見、暗黙の業務ルールといった暗黙知を形式知化し、AI のコンテキストとして整備する作業が必要である。この作業は有識者にしか行えないため、有識者の時間をコンテキスト整備に優先的に配分する組織的な判断が求められる。

## Technology: 既存資産のコンテキスト整備

ブラウンフィールド開発のパターンは、コンテキストエンジニアリングの観点から統一的に整理できる。図8に示す通り、各パターンの違いは、既存システムのどの資産が AI にとっての主要コンテキスト（Primary Context）となるかの違いである。

- 保守・機能追加では、要件、設計、実装、デプロイのすべてが主要コンテキストとなる。既存資産の品質がそのまま AI の出力品質に影響するため、ドキュメントとコードの整合性維持が特に重要となる。
- リホストでは、既存のデプロイ仕様とコードが 主要コンテキストとなる。アプリケーション自体は変更しないため、環境依存の設定や依存関係の把握が成否を分ける。
- リライトでは、既存のコードと設計が主要コンテキストとなる。AI は既存コードから振る舞いを理解し、設計から構造を把握した上で、新しい実装を生成する。
- リビルドでは、既存の要件と設計が 主要コンテキスト となる。設計から実装まで全面的に再構築するため要件の抽出精度が成否を決定するが、既存の設計情報もアーキテクチャ上の制約や過去の設計判断の背景を把握する手掛かりとなり、再設計の出発点として活用される。

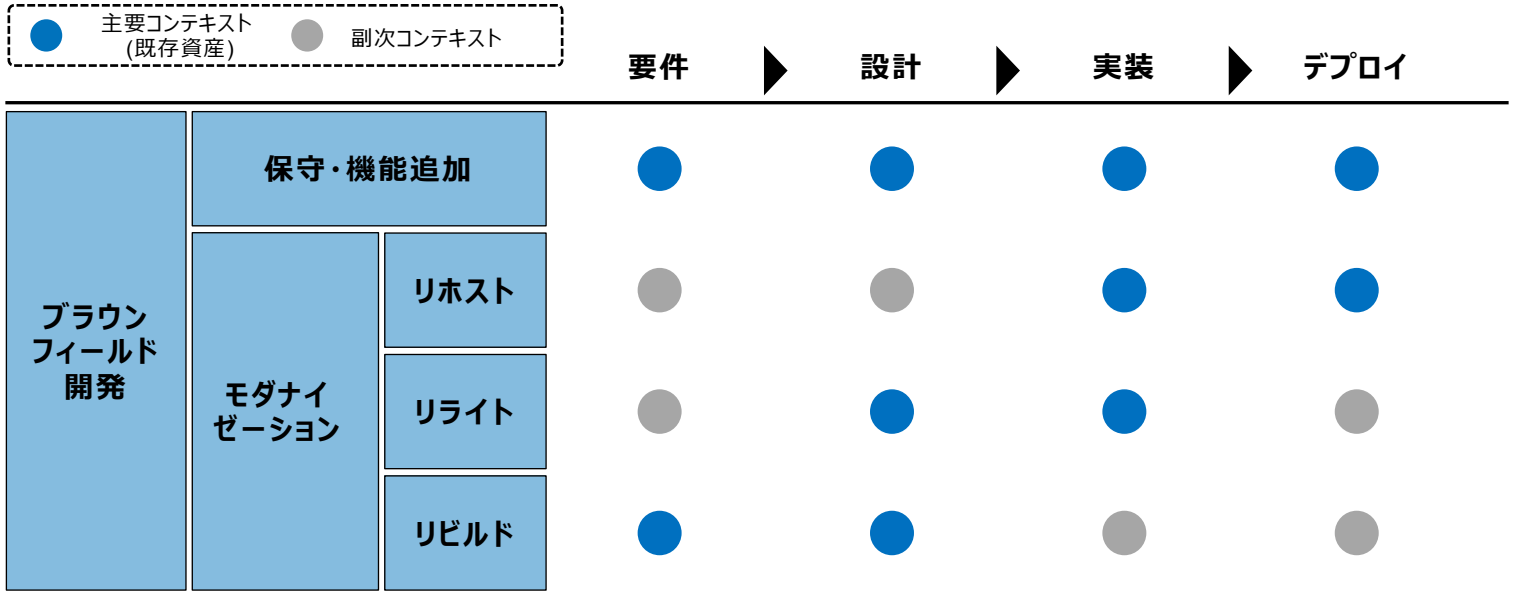



**図8. コンテキストエンジニアリングによる開発パターンの統一的な見方**

既存資産を AI が参照・活用できる形に整備するにあたり、整備の対象は情報資産と開発インフラの双方にわたる。しかし、現実には要件定義書や設計書が不完全であったり、実装と乖離していたりするケースが少なくない。情報資産の整備には主に三つのアプローチがある。

特に保守・運用の文脈では、欠落したドキュメント、更新されていない設計書、退職者しか知らない運用知識といった問題が AI 活用の障害になりやすい。MCP や検索基盤は既存資産の読解を助けるが、それだけで欠落知識が埋まるわけではないため、有識者へのヒアリングや運用手順の再文書化を並行して進める必要がある。

### 1. 既存成果物の変換：

Excel 形式の設計書を Markdown に変換する MarkItDown [10] のような変換ツールや、それらを呼び出す MCP サーバ、マルチモーダル LLM を活用した図表のテキスト化により、既存資産を AI が参照可能な形式に変換する。ただし、このアプローチだけでは人間の開発者が持つ暗黙知（設計判断の背景、過去の障害対応の知見、チーム固有の慣習など）は反映されない。

### 2. 有識者による継続的な情報補完：

AI エージェントが作業を進める中で判断に迷う箇所や期待と異なる出力が生成される箇所を特定し、必要なコンテキストを追記していく。これにより暗黙知が徐々に形式知化され、AI の作業精度が向上する。

### 3. ソースコードからのリバースエンジニアリング：

設計書や要件定義書が存在しない、あるいは実装と乖離している場合に、AI を活用してソースコードから設計情報や要件を復元する。これにより、ドキュメントが不十分なブラウンフィールド環境でもリッチなコンテキストを構築できる。

開発インフラの整備も同様に重要である。

### ■ テストの自動化:

手動テストを自動化し、AI がコード生成・テスト実行・結果確認のループを自律的に回せるようにする。

### ■ 環境構築のコード化：

デプロイ手順や環境構築手順を Infrastructure as Code として整備し、AI が環境操作を自律的に実行できるようにする。

### ■ 連携仕様の文書化：

既存システム間のインターフェース仕様やデータフォーマットを文書化し、AI が既存システムと正しく連携するための前提知識を提供する。

これらの整備作業は一時的なコストを伴うが、AI-Native 開発への移行を可能にするだけでなく、人間の開発者にとっても開発効率と保守性の向上につながる投資である。

# Level 2: AI-Verified — AIが品質を作り込み、人が透明性を担う

前章では、AI が開発実務を担い、人間が品質を作り込む Level 1 のデザインを示した。本章では、前章で述べた移行条件（組織的信頼の蓄積、品質基準の形式知化、検証基盤の整備）を前提に、成熟度モデルの第二段階のデザインを示す。技術的には実現可能な領域に入りつつあり、NTT DATA の実践においても AI によるコードレビューが人間のレビューと同等以上に的確な指摘を行うケースが確認されている。一方で、AI による品質判断の組織的な信頼性の担保や社会的受容など、実用化に向けた課題が残されている。

## この段階の特徴

Level 1 の構造的な限界は、人間のレビューがスループットのボトルネックとなる点にあった。 Level 2 では、品質の監督そのものを AI エージェントに委ねることでこの限界を解消する。AI がテスト生成・実行、コードレビュー、品質モニタリング、品質ゲート判定などの監督活動を自律的に行い、人間はプロセス全体の管理とアカウンタビリティに集中する。人間の介入は、重大な意思決定ポイントや品質基準の逸脱といった状況に集約される。

この段階の鍵となる実現要因は、エージェント型監督である。開発タスクを遂行する執行エージェントと、その成果物を検証する監督エージェントを分離し、監督の主体が人間から AI へと移行する。監督活動の総量が減るわけではなく、変化するのは担い手である。

生産性の観点では、監督も AI が担うことで、生産性の向上が「個人の生産量の増大」から「チーム全体のスケーラビリティの獲得」へと質的に変化する。複数の AI エージェントによる並列開発が本格的に機能し、エージェント数に応じてスループットが水平にスケールする。例えば、10 人がそれぞれ1つの AI エージェントを監督する Level 1 と、同じ 10 人が数十のエージェント群を管理する Level 2 とでは、チーム全体のアウトプット量に大きな差が生じる。

図9に、Level 2 における人間と AI の協働プロセスと主要なイネーブラーの概要を示す。以降の節では、Process、People、Technology の順に各要素のデザインを論じた上で、この段階の課題を整理する。

図9は、執行エージェントと監督エージェントが並列に動作し、人間の関与が要件定義と意思決定ポイントおよび最終確認に集約される構造を示している。

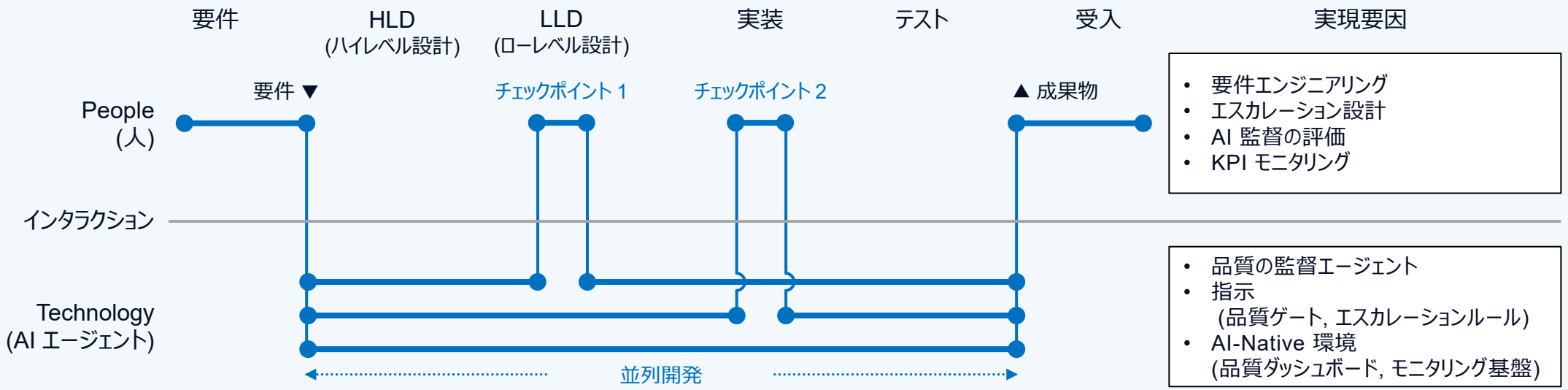



**図9. Level 2 における協働プロセスの全体像**

## Process: Shift-Left — 要件定義への集中

前節で述べた通り、Level 2 では品質管理を AI が監督として担い、人間はプロセス全体を管理する立場へと変化する。この品質確保主体の移行に伴い、人間の役割は開発プロセスの各工程への介入から、品質活動をより上流へ移す Shift-Left、すなわち要件定義への集中へと構造的にシフトする。Level 1 でも上流工程でハイレベルな方針を先に確定させることは有効なアプローチであったが、Level 2 ではこの Shift-Left が構造的な必然となる。その理由は、フィードバックループの粒度の変化にある。

Level 1 では、設計書・コード・テストといった各成果物に対して人間がレビューで介入する フィードバックループが存在し、要件の曖昧さや誤りが下流で発見されても軌道修正できた。レビューが「要件の不備を補完するセーフティネット」として機能していたのである。

Level 2 では、AI が自律的に執行・監督を行うため、成果物ごとの人間のレビューは不要となる。代わりに、AI が重大な意思決定ポイント（アーキテクチャ上のトレードオフ、要件解釈の方針選択、想定外のリスクの検出など）に遭遇した際に人間（マネジメント）にレポートし、判断を仰ぐフィードバックループが構造の中心となる。フィードバックが消失するのではなく、その粒度が「成果物ごと」から「意思決定ポイントごと」へと変化する。

具体的には、Level 1 では各成果物（設計書・コード・テスト）に対して人間がレビューで介入していたのに対し、Level 2 では人間の介入が「入力（要件定義）」「重大な意思決定ポイント（AI からのレポート）」「出力の最終確認」の三点に集約される。このうち前二者は人間のコンテキストを開発プロセスに注入する行為であり、最終確認は成果物の受入判定である（図10）。この変化により人間の介入頻度は大幅に減少する一方、入力の品質が出力の品質を大きく左右する構造となる。

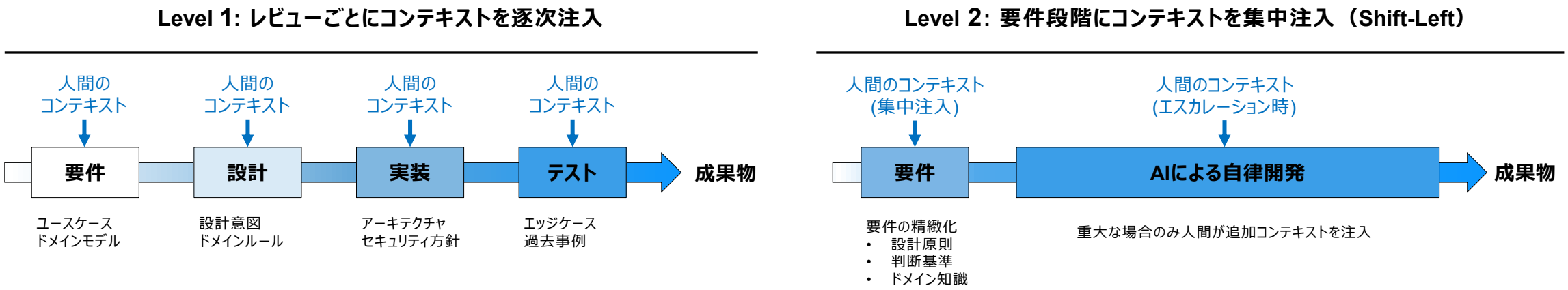



**図10.フィードバックループの粒度変化 — コンテキスト注入の視点**

この構造的変化により、要件に求められる精度・網羅性・検証可能性が格段に高まる。Level 1 では人間がレビューでコンテキストを補完できたため要件にある程度の曖昧さは許容されたが、Level 2 では人間が不備を検出・修正する機会が大幅に限られる。デザインのフレームワーク章で述べたコンテキスト注入の視点でいえば、Level 1 で各チェックポイントを通じて逐次注入していた人間のコンテキストを、Level 2 では要件定義の段階に集中投入する構造への移行である。AI-Native 開発における Shift-Left の本質は、このコンテキストの前倒し集中投入にある（図10）。AI が要件を解釈して自律的に実行するため、人間による要件の明確化に加え、AI を活用した要件品質の体系的な向上が有効となる。例えば、AI が要件間の矛盾や曖昧な記述を検出する曖昧性分 析、要件の網羅性を既存の品質特性や過去の知見に照らして検証する網羅性チェック、要件から想定されるエッジケースやリスクシナリオを自動的に導出するリスク分析、さらには要件からテストケースのドラフトを生成し検証可能性を事前に確認するテスタビリティ検証などが考えられる。これらの取り組みにより、下流工程での手戻りを大幅に削減できる。

また、AI がどのような状況を「重大な意思決定ポイント」として認識し人間にレポートすべきかの基準設計も、要件定義の一部として重要となる。この基準が適切に設計されることで、意思決定ポイントごとのフィードバックループが下流における最後のセーフティネットとして機能する。

## People: レビュアーからプロセス管理へ

Level 2 における人間の役割は、Level 1 の「AI の成果物をレビューし品質を作り込む」から「AIのプロセス全体を管理しアカウンタビリティを果たす」へと変化する。この変化は、人間が個々の成果物に逐一関与する度合いが減り、プロセスの設計・運用・評価というマネジメント活動に集中することを意味する。

マネジメントとしての人間は、AI の執行・監督活動が適切に機能しているかをプロセスレベルで管理する。この管理は、プロセスの可視化と評価、および介入メカニズムの設計という二つの軸で構成される。

第一に、プロセスの可視化と評価である。品質ダッシュボードを通じて、各エージェントの活動状況・品質指標の推移・要件の充足度を継続的に把握し、計画からの乖離を早期に検知する。すべてのコードを逐一レビューするのではなく、集約された品質指標に基づいてプロセス全体を管理する。加えて、AI による品質判定そのものの有効性も評価対象とする。品質ゲート通過率や欠陥検出率などの KPI を設定し、定期的に AI の判定結果と人間の判定結果を比較することで、AI 監督の精度を継続的に検証する。

第二に、介入メカニズムの設計である。AI がどのような状況を人間に報告すべきかの基準を設計し、報告を受けた際の判断プロセス（報告内容の妥当性の確認、選択肢の評価、判断の記録）を定義する。特に、品質基準を満たさない成果物の検出、AI エージェント間の判断の矛盾、想定外の技術的制約の発見など、AI の自律的な判断では解決が困難な状況に対しては、エスカレーションの 報告ルートと対応プロセスを事前に定義しておく。

これらの判断記録は、ガバナンス体がステークホルダーに対してアカウンタビリティを果たす際の根拠となる。

ガバナンス体としての人間は、ステークホルダーへのアカウンタビリティを果たす。Level 2 では人間が直接確認していない工程が増えるため、説明の根拠が変化する。Level 1 では「人間が成果物をレビューした結果」に基づいてアカウンタビリティを果たしていたが、Level 2 では「AI による品質監督の仕組みが適切に機能していること」と「その結果としてプロダクトが要件を満たしていること」の双方に基づく説明が求められる。すなわち、プロセスの有効性とプロダクトの品質という二つの根拠を示すことで、人間が直接確認していない工程に対してもアカウンタビリティを果たす構造である。

## Technology: マルチエージェントによる監督

Level 2 における Technology の核心は、監督を担うマルチエージェントアーキテクチャの設計である。Level 1 では単一の AI エージェントが開発者と対話しながら開発を進める形態が基本であったが、Level 2 では複数の AI エージェントが異なる役割を持ち、協調して品質を監督する構造を構築する。独立した監督エージェントが品質のセーフティネットとして機能するため、デザインのフレームワーク章で述べた透明性と自律性の連続的な段階において、執行エージェントにより高い自律性を付与できるようになる。

監督エージェントの役割は、人間が担ってきた監督活動を AI エージェントの責務として再定義したものである。

1. テスト生成・実行エージェント
   要件に基づくテストケースの生成と実行結果の検証
2. コードレビューエージェント
   コーディング規約への準拠、ロジックの誤り、保守性の評価
3. 品質モニタリングエージェント
   品質指標の継続的な監視と品質ゲートの判定
4. セキュリティ検証エージェント
   脆弱性スキャンや攻撃シナリオのシミュレーション
5. 要件整合性検証エージェント
   実装内容が要件・受入条件と矛盾していないかの確認
6. 依存関係・コンプライアンス検証エージェント —外部ライブラリ、ライセンス、社内標準への適合確認

これらが並列に実行されることで、人間が逐一確認するよりも広範かつ高速な品質監督が可能になる。ただし、スリーラインモデルにおいて第二ラインが第一ラインから独立した立場で監督を行うのと同様に、監督エージェントは執行エージェントから独立して動作させなければならない。生成と検証を同一エージェントが担うと自己バイアスが生じるリスクがあるため、異なるインスタン スとして分離し、独立した指示と判断基準を与える。これらのエージェントの協調には、タスク配分・進捗追跡・結果集約を統括するオーケストレーターが必要となる。

指示と環境の設計も Level 1 から拡張される。指示の面では、Level 1 のコーディング規約やテスト方針に加え、品質ゲートの判定ルール、異常検知時のエスカレーション基準など、監督活動を統制するための指示が必要となる。特に、AI がどのような状況を「重大な意思決定ポイント」と認識し人間にレポートすべきかの基準設計が、人間による管理を形骸化させないための鍵となる。環境の面では、品質ダッシュボード（テストカバレッジ、欠陥検出率、品質ゲート通過率等の集約）、品質指標のモニタリング基盤、エスカレーション時に AI の判断根拠を確認できる仕組みなど、監視インフラの整備が求められる。

## この段階の課題

Level 2 の実現に向けては、構造的・運用的・社会技術的な課題が残されている。

第一に、AI が執行と監督の双方を担うことに起因する構造的なリスクである。例えば、AI がテストの失敗に直面した際に、コードのバグを修正するのではなくテストケース自体を改変して通過させるといった行動が NTT DATA の実践において観察されている。これは、AI が「テストを通す」という目標を最短経路で達成しようとする結果として生じるものであり、品質の監督が実質的に無効化される深刻な問題である。執行エージェントと監督エージェントの分離、テストケースの変更に対する独立した承認プロセスの導入、あるいはテストと実装の変更を同時に行うことを制約として禁止するガードレールの設計など、AI の自律的な品質監督が自己完結的に形骸化しないための仕組みが不可欠である。

第二に、人間の管理の適切な水準という運用的課題である。AI の自律性を高めすぎると人間の管理が形骸化し、問題の発見が遅れるリスクがある。一方、管理を厳格にしすぎると Level 1 と同様のボトルネックが生じ、Level 2 への移行の意義が失われる。この均衡点の設定は、プロジェクトの特性やリスク許容度に応じた判断が求められる。

第三に、AI による品質判断への信頼の確立である。AI の品質判定を技術的に担保できるかという課題と、その判断を組織として受容できるかという課題が表裏一体で存在する。

Roychoudhuryらが論じた「信頼の確立（programming with trust）」[4] はまさにこの段階で中心的な問いとなる。Level 1 では人間が AI を過度に信頼するリスクが課題であったが、Level 2 では逆に、AIの品質判断を組織としてどこまで信頼するかが問われる。Level 1 の実践を通じて精度を測定・評価し、段階的に委譲範囲を拡大するアプローチが現実的である。

これらの課題の根本的な解消は、Level 3 で AI が透明性を自ら提供することで実現される。Level 3 への移行は、Level 2 の実践を通じて以下の条件が整うことで可能になる。

1. 来歴追跡と検証可能性の技術基盤が確立され、AI の判断過程を事後的に追跡・検証できる状態にあること。

2. AI の自己説明の信頼性が実証されていること。Level 2 の運用を通じて、AI が提供する説明と実際の活動の整合性が継続的に検証され、十分な精度が確認されている必要がある。

3. 事後的な監査によってアカウンタビリティを果たすことが、ステークホルダーおよび社会的に受容されていること。

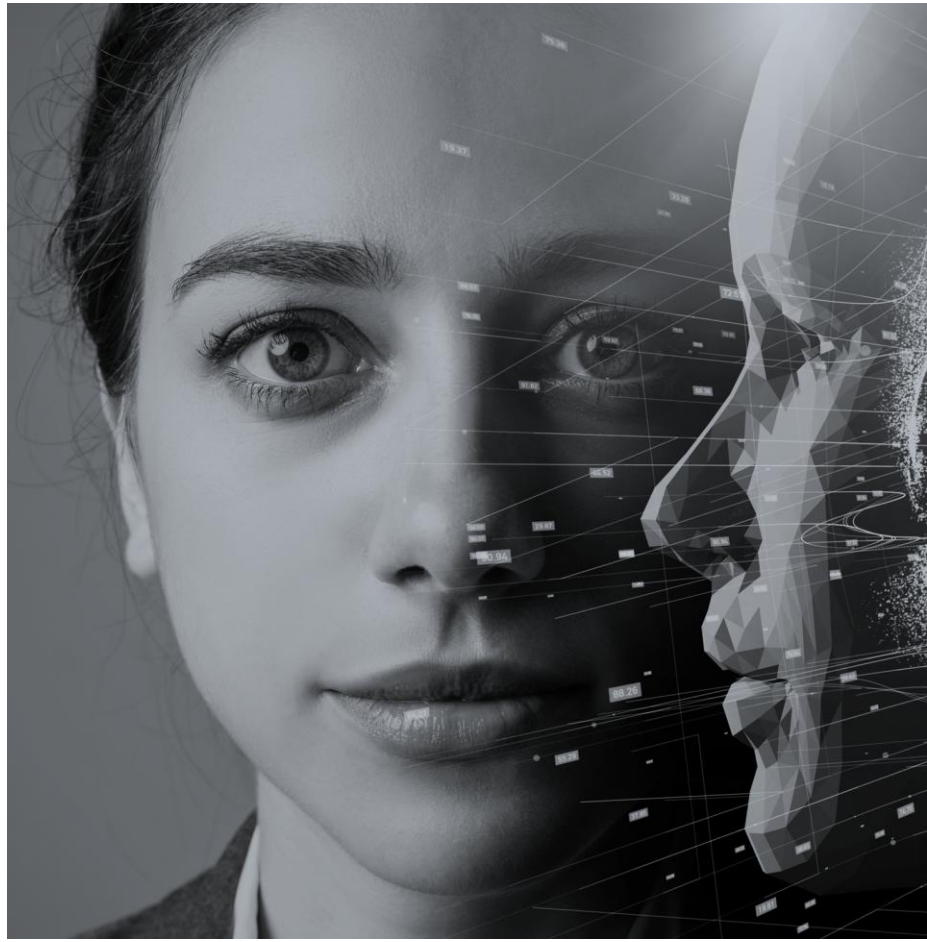
# Level 3: AI-Explainable — AIが透明性を担い、人が説明する

前章では、AI が執行と監督を担い、人間がプロセス管理とアカウンタビリティを担う Level 2 のデザインを示した。本章では、前章で述べた移行条件を前提に、成熟度モデルの第三段階として AI が透明性も自ら提供する段階を展望する。この段階は、直近の導入対象というより AI-Native開発の長期的な到達点として位置づけられる将来像であり、技術的にも社会的にも多くの課題が残されている。

## この段階の特徴

Level 2 の課題で述べた通り、AI の活動規模が拡大するにつれ、人間によるプロセス管理が透明性確保のボトルネックとなりうる。Level 3 は、AI 自身が透明性を提供することでこの限界を解消する段階である。

この段階では、AI が開発プロセス全体の透明性を自ら確保する。なぜその設計判断を行ったの　か、なぜその実装方針を選択したのか、どのようなテスト戦略でどのような品質を確認したのかを、人間が理解・検証可能な形で記録し提示する。「AI-Explainable」という名称は、AI が自らの開発プロセスを説明可能な形で提示できるようになることを意味している。AI がプロダクトとその開発過程の監査証跡（Audit Trail）を合わせて提供するため、人間の関与は事後的な監査活動に集約される。結果として、プロセス管理の負荷が AI に移行し、少人数の監査チームが大規模な AI 開発活動を統制する構造が成立する。



この段階の鍵となる実現要因は、来歴追跡と検証可能性（Lineage & Verifiability）である。AI がどのような入力・コンテキストに基づいてどのような判断を行い、どのような成果物を生成したのかを、追跡・検証可能な形で記録する技術の確立が前提となる。図11に、Level 3 における人間と AI の協働プロセスと主要なイネーブラーの概要を示す。以降の節では、図11の各要素について Process、People、Technology の順にデザインを論じた上で、実現に向けた課題を整理する。

図11は、AI が開発からプロセス管理・透明性確保まで自律的に遂行し、人間の関与が事後的な監査判断に集約される構造を示している。


**図11. Level 3 における協働プロセスの全体像**

## Process: Shift-Right — 事後的な監査

Level 3 では、人間の関与のタイミングが開発プロセスの右側、すなわちプロダクトの完成後・受け入れの段階へとシフトする（Shift-Right）。Level 1 では人間が成果物ごとのレビューを通じてプロセス全体に介在し、Level 2 では人間が要件定義に集中しつつ意思決定ポイントで介入していた。Level 3 では、AI が執行・監督・透明性の確保まで自律的に行うため、人間の主な関与は事後的な監査（Audit）として行われる。これは、デザインのフレームワーク章で述べた透明性と自律性の連続的な段階において、事後検証（by audit）がプロセス全体の基本原則となる段階を意味する。

事後的な監査プロセスは以下の流れで行われる。AI がプロダクトと監査証跡を提出し、人間がプ ロセス品質とプロダクト品質の双方を検証した上で監査判断を下す。プロセス品質の検証では、監査証跡に含まれる来歴追跡の記録から、AI の開発活動の中から統計的にサンプルを抽出し判断の妥当性を確認する（サンプリング監査）。また、AI が提供する判断理由と実際の活動履歴の整合性を検証する（自己説明の整合性評価）。プロダクト品質の検証では、AI が生成・検証した成果物に対して独立した観点からテストを設計・実行する（独立テスト）。監査時に不明な点があれば、AI に対して追加の説明を求めることも可能である。

AI はプロセス管理（スケジューリング、進捗追跡、リスク監視）も自律的に担う。Level 2 では人間がこれらの活動を行っていたが、Level 3 では AI がプロジェクト全体の工程管理、各エージェントへのタスク配分と進捗の追跡、品質指標に基づくリスクの監視と対応を自律的に遂行する。人間はこれらのプロセス管理の結果も事後的に検証する。なお、Shift-Right の実効性は要件段階のコンテキストの充実度に依存する。Level 3 では Level 2 の意思決定ポイントでのセーフティネットも無くなるため、要件が明確でコンテキストが豊富であるほど AI の判断精度は上がり、事後的な監査の負荷も下がる。Level 2 で導入した Shift-Left の深化が引き続き重要である。

## People: 監査者としての人間

Level 3 における人間の役割は、Level 2 の「AI のプロセスを管理する」から「AI の活動を監査する」へと移行する。Level 2 では人間が AI の活動を日常的に管理していたが、Level 3 では AI が自ら透明性を確保するため、人間は必要に応じた独立検証に集中する。

前節で述べた監査プロセス（サンプリング監査・整合性評価・独立テスト）の各活動には、固有のスキルが求められる。サンプリング監査には統計的なサンプリング手法の知識が、整合性評価には AI の自己説明を批判的に分析する能力が、独立テストには AI の検証とは異なる観点からテストを設計する能力がそれぞれ必要となる。加えて、監査結果をステークホルダーに対してわかりやすく伝えるコミュニケーション能力も重要である。Level 2 のプロセス管理スキルから、監査の専門性への転換が求められる。

監査活動の負荷は、AI の自己説明の信頼性に依存する。AI の自己説明が十分に信頼できる段階では、検証すべき範囲が絞られ監査活動は軽量な形となりうる。一方、自己説明の信頼性が十分に確立されていない段階では、より広範な検証が必要となる。監査活動自体にも AI を活用することが考えられるが、最終的な判断は人間が行う。

ガバナンス体としてのアカウンタビリティは、AI の透明性記録と人間の監査結果の双方に基づいて果たされる。AI が提供する来歴追跡の記録が説明の基礎データとなり、人間による独立した監査がその信頼性を裏づける。ガバナンス体は、AI の自己説明を単に信頼するのではなく、監査活動の結果に基づいてステークホルダーに対して説明・正当化を行う。また、スリーラインモデルにおける第三ラインの機能には保証だけでなく助言も含まれる。監査者は検証を通じて得られた知見に基づき、監査基準の見直しや監督プロセスの改善といった AI ガバナンスの向上をガバナンス体に助言する。

## Technology: 来歴追跡と検証可能性の基盤

Level 3 における Technology の核心は、AI の開発活動を事後的に検証可能にするための透明性基盤の構築である。Level 2 で整備した監視インフラ（品質ダッシュボード、モニタリング基盤）を土台として、来歴追跡・検証可能性・監査ツールの三層で構成される。

来歴追跡（Lineage）基盤は、AI エージェントの判断過程と成果物の関係を構造化して記録する仕組みである。記録の基本構造は、監査の観点である IPO（Input, Process, Output）に従う。すなわち、各判断における入力（参照した要件・コンテキストとそのバージョン）、処理過程（検討した選択肢と選択理由）、出力（生成された成果物とその変更差分）の関係を構造化データとして記録し、任意の成果物からその生成に至った判断の連鎖を遡及できる状態を目指す。この来歴追跡の発想は、ソフトウェアサプライチェーンにおける SBOM（Software Bill of Materials）や SLSA（Supply-chain Levels for Software Artifacts）と類似している。これらが「どのコンポーネントからビルドされたか」を追跡するのに対し、Level 3 の来歴追跡は「どのコンテキストと判断からコードが生成されたか」を追跡する。記録フォーマットの標準化や検証の自動化といった技術的知見は参考になる。

検証可能性（Verifiability）の仕組みは、個々の判断の妥当性を事後的に評価可能にする基盤である。LLM はサンプリングやコンテキストの微細な差異により実用上は非決定的な振る舞いを示すため、同一の入力から同一の出力を再現することは現実的ではない。重要なのは出力の再現ではなく、判断の妥当性の検証である。各判断時点での入力・コンテキスト・制約条件を構造化して記録し、「このコンテキストであればこの判断は妥当であったか」を第三者が評価できる状態を目指 す。ここで、AI が自らの活動の監査証跡を自ら生成するという構造上、記録の正確性と改ざん耐性の担保が固有の課題となる。来歴追跡記録への暗号学的署名やハッシュチェーンによる改ざん検　知、あるいは記録プロセス自体を独立したエージェントに分離するといった設計が考えられる。

監査ツールは、蓄積された透明性データに対して人間が効率的にクエリを実行し、検証を行うためのインターフェースである。特定の成果物がどのような判断過程を経て生成されたかの追跡、品質指標の異常値の原因分析、AI の自己説明と実際の活動記録の整合性検証などの機能を提供する。 Level 2 の品質ダッシュボードがリアルタイムのモニタリングに主眼を置くのに対し、監査ツールは判断の連鎖の遡及的追跡に主眼を置く点で異なるが、データ基盤は共通化できる。

## 実現に向けた課題

Level 3 の実現には、技術的・社会的・倫理的な課題が残されている。

技術的には、来歴追跡基盤の確立が最大の課題である。AI エージェントの判断過程は高頻度かつ大量に発生するため、どの粒度で記録すべきかの設計、大量の透明性データの効率的な管理と検索、記録フォーマットの標準化が求められる。既存のソフトウェアサプライチェーンセキュリティ（SBOM、SLSA 等）の知見は参考になるものの、AI の判断過程の記録という新たな対象に適した技術の開発が必要となる。また、Technology 節で述べた通り、AI が自らの監査証跡を生成するという構造に起因する記録の正確性と改ざん耐性の問題も解決が求められる。

社会的には、AI の自己説明を法的・制度的に有効なものとして認める枠組みの整備が必要である。 Ulloa らが指摘する通り、アカウンタビリティは人間以外の主体に委任できない [2]。現在の法制度や業界標準の多くは意思決定の主体が人間であることを前提としており、AI による説明がどの ような条件を満たせば有効とみなされるかの基準はまだ存在しない。この枠組みの整備には、Level 2 における AI による品質保証の社会的受容が先行して進む必要がある。

倫理面では、監査の十分性が最も困難な問いとなる。AI がどこまで自律的に開発を行おうとも、 アカウンタビリティの帰属先はガバナンス体、すなわち人間にある。これは本稿を通じて一貫した前提である。Level 2 では「AI の品質判断をどこまで信頼するか」が問われたが、Level 3 ではさらに踏み込んで「人間がどこまで確認すればアカウンタビリティを果たしたといえるか」が問わ れる。サンプリング監査の範囲、独立テストのカバレッジ、自己説明の整合性評価の深さは、AI の自律性の度合いやシステムのリスク特性に応じた判断が求められ、一律の基準を設けることは難しい。この十分性の判断基準を確立することが、ガバナンス体の最も重要な責務となる。

NTT DATA としては、商用システムの開発において、現時点ではこの段階は射程外である。しかし、AI 技術の進化速度を考慮すると、これらの問いに向き合い、検討を始めることは時期尚早ではない。第一段階、第二段階での実践を積み重ねながら、第三段階への移行可能性を継続的に評価していく姿勢が重要である。

# まとめ

本稿では、アカウンタビリティを軸に AI-Native 開発のデザイン戦略を論じた。IIA のスリーラインモデルをソフトウェア開発に適用し、AI が担う範囲が段階的に拡大する三段階の成熟度モデルを提示した。

Level 1 では AI が開発実務を担い、人間が品質保証・透明性確保・アカウンタビリティのすべてを担う。

Level 2 では AI が品質の監督も担い、人間は要件定義への集中（Shift-Left）とプロセス全体の管理に移行する。

Level 3 では AI がプロダクトとともに監査証跡を提供し、人間は事後的な監査（Shift-Right）によってアカウンタビリティを果たす。

全レベルを通じて変わらない原則は、アカウンタビリティは人間に帰属するということである。AIの役割が拡大しても、ステークホルダーに対する説明・正当化の責務は人間が担い続ける。変化するのは、そのアカウンタビリティを果たすために人間が何に基づくか（自らのレビュー結果か、AIのプロセス管理の結果か、AI が提供する監査証跡に対する監査結果か）という基盤である。AI の自律性が高まるほど、監査の十分性をどう判断するかが重要かつ困難な問いとなる。

実務的には、組織はまず三つの点から着手するのが現実的である。第一に、要件・設計・運用手順を AI が読める形に整備すること。第二に、レビュー観点や品質基準を明文化して、AI と人間の双方が同じ基準で判断できるようにすること。第三に、変更履歴、判断理由、承認記録を残すプロセスを先に整備し、アカウンタビリティを後付けにしないことである。

AI 技術は今後も急速に進化を続ける。重要なのは、特定のプロセスに固執することなく、技術の進化に応じてプロセスを適応させていく姿勢である。本稿で示したフレームワークが、読者各位の AI-Native 開発の一助となれば幸いである。

# 参考文献


1. NTT DATA. "Responsible Software Development in the Age of AI". 2024. https://www.nttdata.com/global/en/-/media/nttdataglobal/1_files/services/application-services/ application-development/responsible-software-development.pdf

2. Ulloa, M., Butler, J. L., Haniyur, S., Miller, C., Amos, B., Sarkar, A., Storey, M.-A. "Product Manager Practices for Delegating Work to Generative AI: 'Accountability must not be delegated to non-human actors'". arXiv:2510.02504, 2025. https://arxiv.org/abs/ 2510.02504

3. Anthropic. "Building effective agents". 2024. https://www.anthropic.com/research/building-effective-agents

4. Roychoudhury, A., Pasareanu, C., Pradel, M., Ray, B. "Agentic AI Software Engineers: Programming with Trust". Communications of the ACM, 2026. arXiv:2502.13767. https:// arxiv.org/abs/2502.13767

5. The Institute of Internal Auditors (IIA). "The IIA's Three Lines Model: An Update of the Three Lines of Defense". 2020. https://www.theiia.org/globalassets/documents/resources/the-iias-three-lines-model-an-update-of-the-three-lines-of-defense-july-2020/ three-lines-model-updated-english.pdf

6. Karpathy, A. "There's a new kind of coding I call 'vibe coding'...". X (formerly Twitter), February 2025. https://x.com/karpathy/status/1886192184808149383

7. Tan, G. "For 25% of the Winter 2025 batch, 95% of lines of code are LLM generated.“ X (formerly Twitter), March 2025. https://x.com/garrytan/status/1897303270311489931

8. Xebia. "2026: The Year Software Engineering Will Become AI Native". 2025. https://xebia.com/news/2026-the-year-software-engineering-will-become-ai-native/

9. ISO/IEC. "ISO/IEC 25010:2023 — Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model".2023. https://www.iso.org/standard/78176.html

10. Microsoft. "MarkItDown — Python tool for converting files and office documents to Markdown". GitHub. https://github.com/microsoft/markitdown



Author

**Keita Takenouchi**

Technology Strategist
(Software Engineering)



Author

**Shungo Masaki**

Technology Specialist
(AI & Software Engineering)