生成AIとシステムの境界線を見極める

プロダクト検証で素早く確かめ、リリースで生成AI・システム・人の境界線を引き直す

Consultant
Yuki Yamashita

Red Hat

その AI Agent、本当に「そのまま」で良いか

AI Agent 開発の相談が増えるいま、本資料が最初に立てる問い

1

相談が増えている

「AI Agent で実現したい」という相談が、業種・規模を問わず増えている。前向きな流れ。

2

設計を見ると、引っかかる

アーキテクチャを見せてもらうと「ここは AI に任せるより、製品やシステムで作るほうが良いのでは」と感じる箇所がある。そう指摘する。

3

返ってくるのは、同じ答え

「すでに動いているので、そこは変えたくない」。前向きな相談ほど、この一辺倒な反応になりやすい。

―― 本当に、そのままで良いのか？

この問いを言語化したのが本資料。「どこまで AI に任せ、どこからシステムで固めるか」を、これから一緒に見極める。

「そのままで良いか」を分解すると、3つの論点に絞れる

ここでは特に大事な3つに絞る（細部は後のスライドで深掘り）

1 「AIで済むなら安い」は本当か？

見かけ上は安い。でも、使い続けても本当に安いのか。

2 品質・正確性は保てるか？

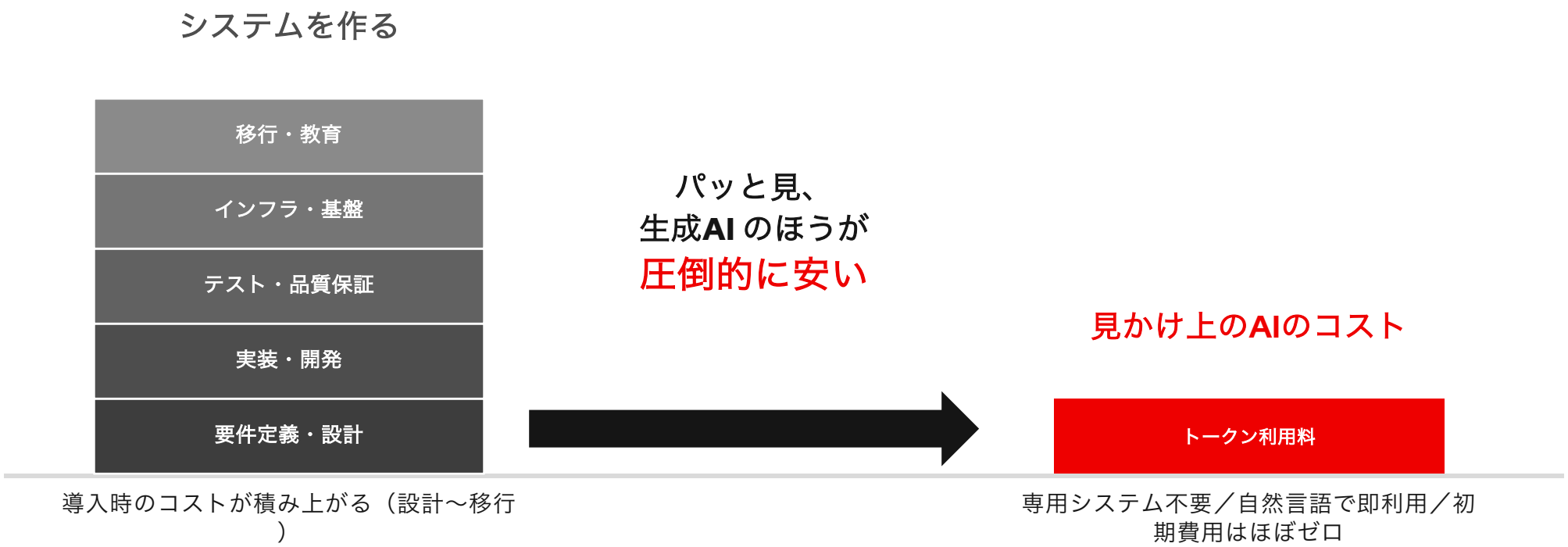
もっともらしく間違える。事実や数字を任せて事故にならないか。

3 どこまでAIに任せるか？

誰が結果に責任を持つか。そして、プロンプト／スキル／スクリプト／システム――どれで作るのが正解か。

これらを順に紐解き、「どこまでAIに任せ、どこからシステムで固めるか」の判断軸を持ち帰ってもらう。

「システムを作る」より「AI に聞く」ほうが、一見ずっと安い



ーー本当にそうか？　初期費用ではなく「運用に入った後のコスト」で見る必要がある（次ページ）

「AI で済むなら安い」は本当に正しいか

見かけ上のAIのコスト

トークン利用料

専用システム不要／自然言語で即答／PoC・単発作業は速い

実際に使い続けると

本当は見えていない生成AIのコスト

- 監査・認証・ログ
- 機密データの扱い・権限
- 出力の確認・事実チェック
- 入力の準備・コンテキスト整備
- 利用料（トークン・サブスク）

個人利用でも SaaS でも避けられない（形は変わる）

判断軸は「AI の利用料」ではなく、「業務で安全に使い続けるコスト」。

Starbucks – AI 在庫管理ツール、9 ヶ月で撤退

似た見た目の商品を識別できず、店頭でカウントを間違える事故が多発

何が起きたか

2025 年 9 月、北米 11,000 店超に導入された AI 在庫カウント（NomadGo 社、画像認識＋LiDAR、“99% 精度・8 倍速”を謳う）。精度不足で 9 ヶ月後の 2026 年 5 月に撤回。

なぜ起きたか

似た商品（ミルクの種類）の誤認や棚の見落としが頻発。自社の宣伝動画ですら誤検知。“99% 精度”はマス展開前に独立検証されていなかった。

境界線を見極める

「在庫の正確な数」＝信頼できるデータ を AI に任せた。ニアパーフェクトな精度・再現性が要る領域は、AI の苦手分野。

教訓

確認が必要な AI は効率化にならない––全出力を人が検算するなら作業は二重になる。在庫 DB・POS を“正”にして AI は補助。ベンダーの精度主張は導入前に検証する。

Deloitte – AI 生成レポートで $290K を返金

オーストラリア政府向け 237 ページ報告書に、存在しない学者の引用・存在しない判例多数

何が起きたか

2024 年末、豪政府がコンプライアンス枠組のレビューを $290K で発注。Deloitte は GPT-4o を使って 237 ページの報告書を作成・納品。研究者が読んで、存在しない学者・存在しない論文が大量に引用されていることを発見。

なぜ起きたか

ハルシネーション (もっともらしい虚偽生成) を、人間がチェックなく承認した。AI は “それらしい引用” を作るのが得意で、生成時点で「本当に存在するか」は保証しない。

境界線を見極める

「事実 (引用の存在)」と「責任 (政府向け公式報告)」を AI に丸投げ。両方とも AI の最苦手分野。誰が事実を担保するかが設計されていなかった。

教訓

AI 出力の事実部分は、必ず信頼できるデータ源 (文献 DB、引用管理ツール) で照合してから人が承認する。

Air Canada – チャットボットの誤案内で敗訴

AI チャットボットが「事後の慶弔運賃申請による返金可能」と回答 – 規定に反する案内で訴訟負け

何が起きたか

2022 年、Jake Moffatt 氏が祖母の葬儀で航空券を購入する際、AI チャットボットに「通常運賃と慶弔運賃の差額は事後申請で返金可能」と案内された。実際の規定は「事前申請のみ」。差額 $812 を求めて提訴、Air Canada が敗訴。

なぜ起きたか

規定 DB（正本）に紐づかず自由に回答を生成し、自社の別ページの規定と矛盾。会社は「ボットは別法人」と弁明も、裁判所は責任は会社にあると判断。

境界線を見極める

問題は“チャットボットを置いたこと”ではない。規定（正＝Source of Truth）に紐づけず AI に自由生成させたこと。事実・契約の案内は AI 単独に任せない。

教訓

①規定を必ず引用する作りにし、正本と矛盾させない。②「サポートに確認を」の注意書きだけでは免責されない（裁判所は“利用者が確認すべき”を退け、会社の責任とした）。

3 つの失敗事例に共通する、境界線の引き間違い

Starbucks

AI を「数の“正解”」にした

何を任せたか
在庫の正確な数（＝“正解”）を AI に出させた

結果
ニアパーフェクトな精度が出ず、確認が二度手間で撤退

Deloitte

AI に「事実そのもの」を作らせた

何を任せたか
引用・事実を AI に生成させ、裏取りなしで承認

結果
存在しない引用が多数、**$290K** を返金

Air Canada

AI に「規定の回答」を任せた

何を任せたか
規定の案内を、正本に紐づけず AI に自由生成させた

結果
自社の別ページと矛盾し、会社が敗訴

共通点：3 社とも AI を「事実・数値・規定の “正解そのもの” を出す担い手」にした。
AI は “意味づけ（解釈・要約・言語化）” は得意でも、“正解” は保証できない。正解はシステム／人に置き、AI は橋渡しに限る――その線を引いていなかった。

生成AIと人とシステムの役割を考える

人を消すのではなく、判断を高い場所へ移す — タスクごとに役割を分解して再配置する

人

判断者・監督者・改善者

- 最終判断と承認
- 例外対応
- 改善方針の決定
- 顧客向けの判断

生成AI

解釈・要約・仮説(非定型タスク)

- ログ要約
- 手順書検索
- 過去事例検索
- 一次仮説の提示
- 報告書ドラフト

システム / スクリプト

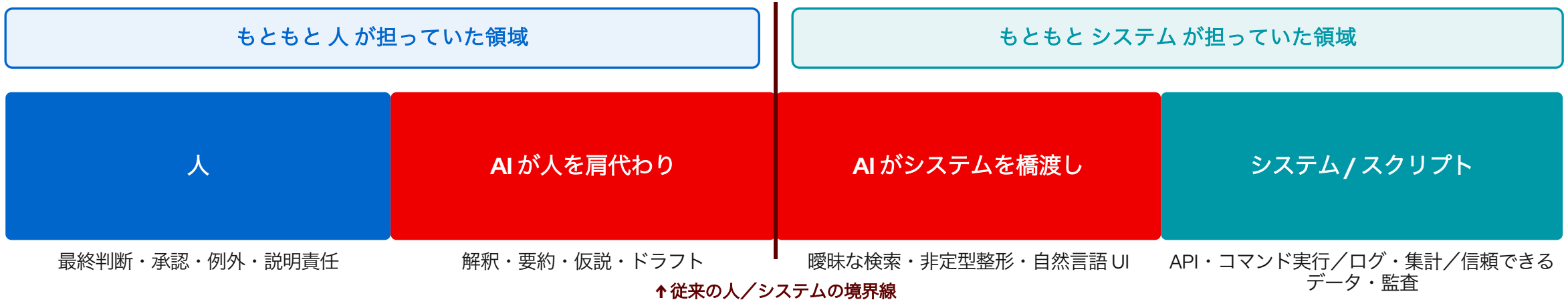
正確・頻繁な実行(定型タスク)

- API/コマンド実行
- ログ取得
- データ集計
- 通知
- 監査ログ保存

「人を高い判断に移す」設計：作業の繰り返しはAIへ、判断・承認は人に集約する

生成AI のカバー領域は、人とシステムの両方に被さる

従来は人とシステムで分担していた業務に、生成AI が境界をまたいで重なってきた



生成AI は人とシステムの境界に重なり、両者が苦手だった「曖昧・非定型な業務」を肩代わりする。最終判断・信頼できるデータ・監査は人とシステムに残る。

生成AIには、明確な得意分野と苦手分野がある

得意は「意味づけ」、苦手は「信頼性・再現性・責任」

得意なこと

- ◯ 曖昧な指示の解釈
- ◯ 仮説出し
- ◯ 非構造データの理解
- ◯ プロトタイピング
- ◯ 生成・要約
- ◯ 暗黙仕様の言語化

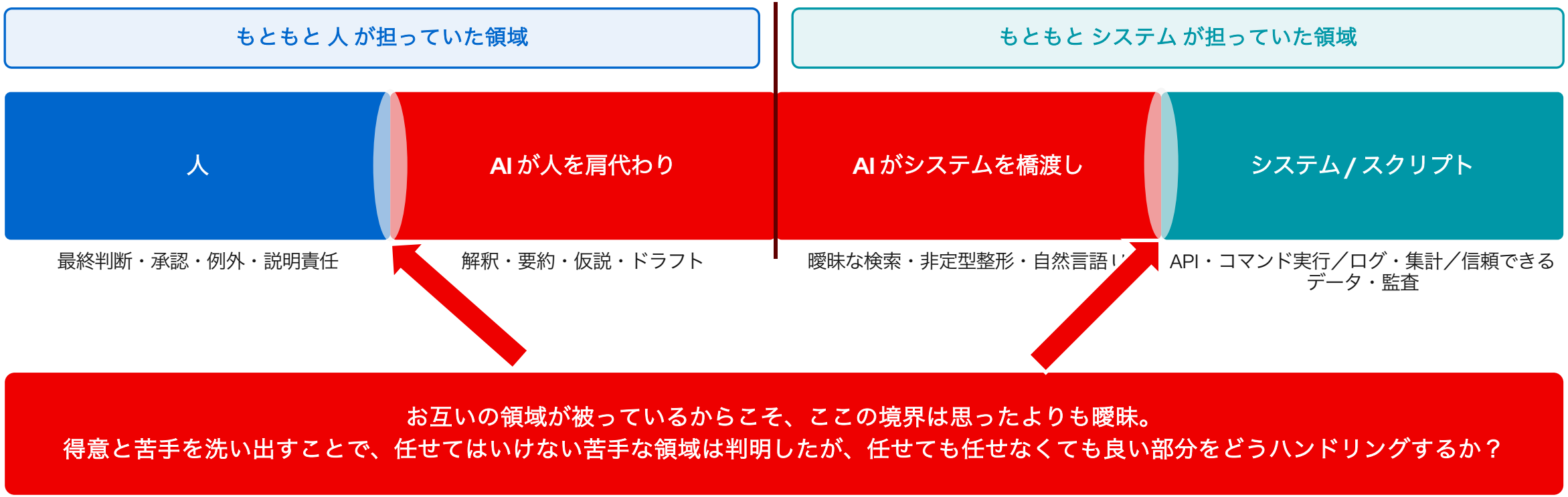
苦手なこと

- △ 同一入力での再現性
- △ 監査・承認・最終判断
- △ 大量・最新データの正確処理
- △ 最新の社内事実(権限/構成/規定)
- △ 長期・横断の一貫性
- △ 厳密な数値の保証

この得意・苦手の輪郭が、境界線の引き方と、“AIが輝く場所”を決める。

生成AI のカバー領域は、人とシステムの両方に被さる

従来は人とシステムで分担していた業務に、生成AI が境界をまたいで重なってきた



生成AIは「PoCを作ること」に圧倒的に向いている

AIの得意分野

曖昧の理解／非構造データ／言語化／プロトタイピング
人とシステムの範囲をある程度、カバーできる

＋

得意・苦手を分類して分かった事実

苦手（再現性・信頼性・責任）な分野は、どれもリリースし、運用しようとした場合に辛い部分であって、UXにはあまり影響がない

→

生成AIが本当に強いのは、
“最初の動くものを作る”場面。

→ PoCを高速で回す道具として位置づけ直す

得意の「プロトタイピング・曖昧の理解・言語化」は、まさにPoCで求められる能力。

得意は“プロトタイプ検証”で効き、苦手は“運用”で辛くなる

得意・苦手を“段階”の軸で並べると、AIが最も輝く場所が見えてくる

段階 →	プロトタイプ検証 （動くものを作る）	プロダクト検証	リリース
得意 速さ・試作・曖昧の理解	◎ 最大に効く	○ 効く	△ 相対的に下がる
苦手 再現性・信頼性・責任	△ まだ出ない	○ 出始める	× リスク大

生成AIのベネフィットが一番得られる領域が“プロトタイプ検証”
その一方リリースに向けて、生成AIの役割を限定し、意味や文脈を扱う情報処理システムとしての位置づけに収めていく。

生成AIの得意分野を活かしつつ運用にも耐える

プロトタイプ検証：まず確かめる

・AI に思い切り任せる

・本来システム化する部分も Agent が肩代わり

・「使われるか」を最短で確かめる

プロダクトリリース：運用に耐える形へ

・AI・システム・人で役割を再分配

・信頼できるデータの貯蔵・頻繁な繰り返し・定型的な作業はシステムへ

・責任の観点で何を任せて何を任せないかの境界線を探る

得意を活かして速く確かめ、その後に運用に耐える形へ作り替える。
AIで高速PoCで開発したプロダクトをリリースするための作業で先程のシステムとの境界線の考えが指針になる。

PRACTICE

考え方から、実践へ

問題と境界線の“考え方”を、現場の手順とツールに落とし込む

これまで：問題と「考え方」（なぜ／何を）

ここから：「実践」（どう作るか・**How**）

境界線を見極める６ステップ

前提：プロトタイプ検証で「使われる」と確証が取れた処理を、運用に耐える形へ作り替えるときに回す。

観る		決める	動かす		続ける
1 観察	2 候補抽出	3 判断	4 設計	5 移行	6 再評価・廃止
最初から仕込む。後付けでは実態（誰が・何回・どこで失敗）が取れない	固める価値がある処理を数字で選ぶ（高頻度・影響大・確認の手間が大きい）	処理ごとに「AIのまま／人の確認付き／システム化／やめる」を決める	AIに任せる範囲・人が確認する箇所・記録の残し方を具体的に決める	一部から段階的にシステムへ置き換える。いつでも前に戻せるようにする	使われ方とコストを定期的に見直し、不要になった処理は止める

観る → 決める → 動かす → 続ける。プロダクト検証で得た学びを運用に耐える役割分担へ変換する。

レイヤー別の技術スタック

構成要素を代表的なソフトウェアに対応づける（生成AI・エージェント周辺を含む。製品は代表例）

レイヤー	担うこと	代表的な技術（OSS・手法）
信頼できるデータ (SoR)	事実・状態・履歴・権限	PostgreSQL／DWH(Snowflake 等)／ServiceNow(ITSM・CMDB)／Okta(IAM)
統制・実行制御 (SoC)	誰が・いつ・何を実行できるか	Argo／Temporal／LangGraph(ワークフロー型)／Tekton／OPA
AI 層（モデル・RAG）	要約・説明・仮説・自然言語 UI	LLM(各社)／ベクタDB(Milvus・pgvector)／RAG
AI エージェント・実行クライアント	スキル/ツールを呼んで実行	LangGraph・deepagents（多段）／Hermes 系（ツール実行）／MCP 対応クライアント(Claude・Goose・Cline)
観測・証跡・評価	LLM/エージェントの記録・評価	Langfuse／LangSmith（LLM トレース・eval）／OpenTelemetry／Prometheus・Grafana
移行・ガードレール	安全に段階移行	Feature Flag(OpenFeature)／カナリアリリース／レート制限
実装手段（速さ↔固さ）	Prompt→Skill→Function→Workflow→System	プロンプト／スキル／一部処理のコード化／ワークフロー(LangGraphなど)

速さが要る層は AI に、固さが要る層はシステムに。技術選定も境界線に従う。

今日から試せる、４つの実装プラクティス

小さく作り、観測し、評価する。手元のエージェントにそのまま適用できる。

1 スキルは「1目的」に割る

大きい skill.md は最後の10%で読み飛ばしが出る。
長いプロンプトを機能単位に分割。各スキルに入力／出力／禁止を明記。

2 決まった確認は「コード」に逃がす

手順を守らせるより、確認自体を Tool/MCP コードに実装。
よく使うチェック3つ（ログ取得・閾値判定・定型API）を関数化。

3 呼び出しを「観測」する

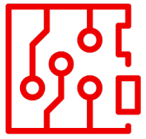
どこで失敗・コスト増かを可視化。勘でなく事実で直す。
Langfuse を入れ、全 LLM 呼び出しに trace を付ける。

4 失敗例を「評価セット」に

代表ケースをテスト化し、変更時に自動で回帰テスト。
直近の失敗10件を回帰テスト化（promptfoo／LangSmith）。

小さく作り、観測し、評価する――境界線を“運用データ”で引くための土台。

まとめ



① AIの使い所

「AI で全部やる」と「AI に任せるな」は、対立ではなく、使い方、使い所が違うだけ。

② プロダクト検証 = 始める速さ

プロダクトの筋を最短で確かめる。捨てやすい構造で、ピボット可能性を最大化する。

③ リリース = 境界線設計

AI・システム・人で役割を引き直し、運用に耐える形にする。

④ 生成AI の性質

得意は “曖昧・非構造・言語化”、不得意は “信頼性・再現性・責任”。

⑤ 6 ステップのフィードバックループ

観察 → 決める → 動かす → 続ける、で境界線を見極める。

⑥ 結論

AI で始め、システムで固め、人が責任を持てる形に設計する。