



生成AIで価値の生み方が変わる

ソフトウェア開発における人材・体制・提供価値を考える

人数 → 判断力

作業 → 設計・検証

文書ベースの提案 → プロトタイプでの提案

はじめに

生成AIの登場により、
仕事の進め方や、価値の置き方は大きく変わり始めています。

これまで人が時間をかけて行っていた
調査・整理・作成・検証といった作業は、
AIによって短時間で実行できるようになりました。

ソフトウェア開発も、その変化の外側にはありません。

作業の価値は圧縮され、
これからは「何を作るか」「どう判断するか」の重要性が高まっていきます。

本資料では、
AI時代における価値観の変化、
求められる人材と体制、
そして私たちが提供できる価値について整理します。

本資料の全体構成

AI時代における開発の前提・人材・提供価値を全25スライドで整理

第1部

AIが変える 価値観の変化

- 01 AIの登場による影響
- 02 AIが変える領域
- 03 工程別の作業量の変化
- 04 AI時代に価値が高まる領域

第2部

AI時代の 人材と体制

- 05 価値が上がる人材
- 06 経験 × AI の効果
- 07 主張の核心
- 08 マンパワー開発の終焉
- 09 開発体制モデル
- 10 若手育成の再設計

第3部

私たちが 提供できる価値

- 11 顧客にとっての価値
- 12 顧客価値を実現する進め方
- 13 自社製品への展開
- 14 AIエージェント対応業務システム
- 15 おわりに

別紙 — 経験差の事例

- ① 要件定義の差
- ② エンティティ設計の差
- ③ コーディング指示の差
- ④ コーディング成果物の差





第 1 部

AIが変える価値観の変化

AIの登場により、開発に求められる価値は変わりつつあります。
ここでは、どのような変化が起きているのかを整理します。

AIの登場で、業界の前提が変わった

生成AIの登場は、単なるツール追加ではありません。

仕事の進め方、成果の出し方、人に求められる役割などが変わり始めています。



作業の圧縮

人手で行っていた作業が短時間で進む



提案スピードの向上

早期に形にして議論できる



顧客期待の上昇

求められる成果の水準が上がる



品質管理の重要性

AI出力の見極めが必要になる



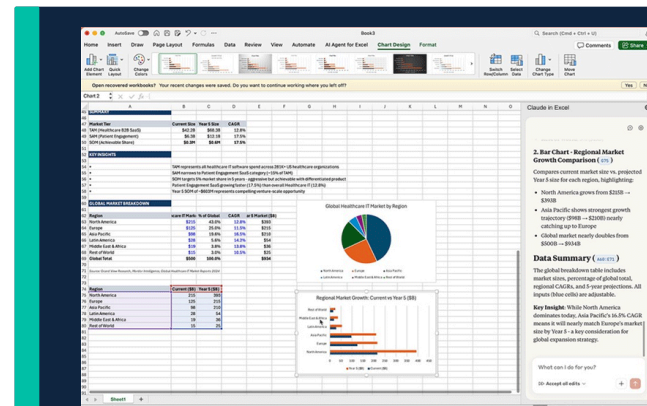
役割の重心の変化

単純作業はAIにやらせる



競争軸の変化

人数より判断力が問われる



実例
業務ツールにもAIエージェントが入り始めている

Excel for Claude

Microsoft Excelを自然言語で操作するAIエージェント

表計算という業務領域そのものに、AIが踏み込み始めている。

これは流行ではなく、開発の前提が確実に変わっていることの証左です。

AIが代替し始めた作業領域

ソフトウェア開発における「調べる」「たたき台を作る」「コードを書く」「テスト観点の洗い出し」
こうした作業は、AIで大きく圧縮されつつあります。
重要なのは、AIの作業と人の作業の線引きを適切に引くことです。

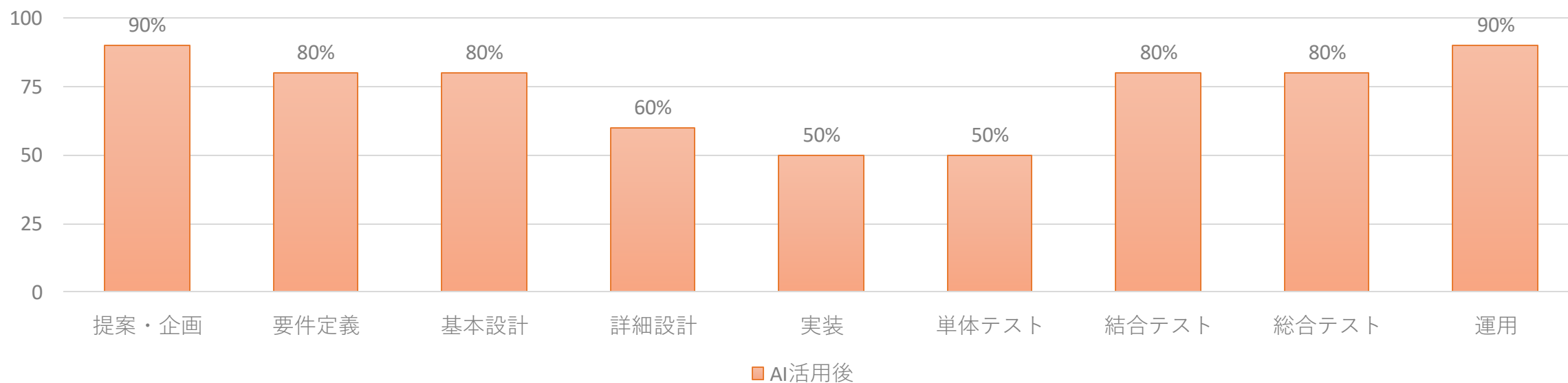
作業領域	従来	AI活用後
技術・業務調査	担当者が調べて整理	AIで候補・論点を短時間で整理
コード作成	人が一から実装	AIがたたき台を生成又は全て作成
テスト観点の洗い出し	経験やチェックリスト依存	AIが観点を広く洗い出し作成・実施
資料作成	手作業で構成・文章化	AIが初稿を作成
仕様整理	議事録やメモの整理	議事録等からAIが要点化・比較表化
リファクタ案	経験者が検討	AIが候補を提示

工程別：AI活用による作業量の変化

AI活用は工程によって効果が大きく異なります。実装やテスト系では大きな圧縮効果が見込める一方、要件定義や運用判断には人間の領域が依然として大きく残ります。

つまり、判断主体である人材が必要となる比率は工程ごとに変わります。

※当社での実績値



大きく圧縮できる

実装・単体テスト・詳細設計

定型生成と網羅的な観点出しが効く工程

一定の圧縮効果

基本設計・結合テスト・総合テスト

たたき台支援はあるが、判断は人が主導

圧縮しにくい

提案・企画 / 要件定義 / 運用判断

業務理解とリスク判断が中心の領域

AI時代に、価値が高まる領域

この様にAIによって、実装・資料作成・調査などの作業量は大きく圧縮されていきます。
一方で、業務の背景を理解し、関係者の意図を整理し、曖昧な要件を現場で使える形に落とし込む力の重要性は、むしろ高まります。
特に、複雑な要件、判断責任、技術選定、プロジェクト推進が絡む領域では、単なる作業量ではなく、何を作るべきかを見極める力が価値になります。

01

システム設計

システムアーキテクチャ設計

非機能要件、技術選定、システム間連携の設計判断。
事業特性とトレードオフを踏まえた設計はAIには委ねられません。

02

業務の理解

複雑な要件定義・合意形成

ステークホルダー間の対立調整、業務観察、暗黙知の言語化。AIに渡す前段階の「要件を作る」仕事です。

03

判断責任

権限・承認・監査・法令対応

職務分掌、内部統制、業界固有の規制対応。判断責任と説明責任、組織ルールの理解が必要為、人間が設計・運用すべき領域です。

04

止められない領域

既存システムからの移行・統合

レガシー解析、データ整合性、段階移行、停止できない業務との両立。販売管理・在庫・会計など複数システム連携も含みます。

05

仕様化しにくい領域

複雑な操作性が求められる画面

業務効率を左右する画面、独自操作、リアルタイム更新などは、仕様書だけでは意図を伝えきれないことがあります。

06

経験が決定的

障害対応・運用保守

想定外の事象では、原因特定・影響判断・関係者対応が同時に必要。AIには判断できない、現場の総合判断力が問われる領域です。

これらの領域では、単なる作業力ではなく、業務理解・設計・品質・リスクを総合的に判断する力が求められます。



第 2 部

AI時代の人材と体制

AI時代に価値が高まる領域を踏まえ、次に、それを担う人材像と開発体制を見ていきます。

価値が上がるのは「判断できる人材」

AIで作業量は圧縮できる。しかし「何を、どこまで、どう作るか」「何を捨てるか」これは依然として人間の判断領域です。
これまで一括りに「経験」と呼ばれていた力は、AI時代には次のような判断力として、より明確に価値を持つようになります。

01 設計力

将来の変更・保守・拡張を見越し、システムの構造や責任分担を決める

02 業務理解力

現場業務・例外処理・運用上の制約を理解し、仕様に落とし込む

03 品質判断力

AI出力や成果物が、業務・保守性・性能の面で妥当かを見抜く

04 リスク判断力

セキュリティ・権限・監査・障害・運用リスクを先回りして見抜く

05 取捨選択力

作るべきもの、作らないもの、後回しにするものを判断する

06 検証力

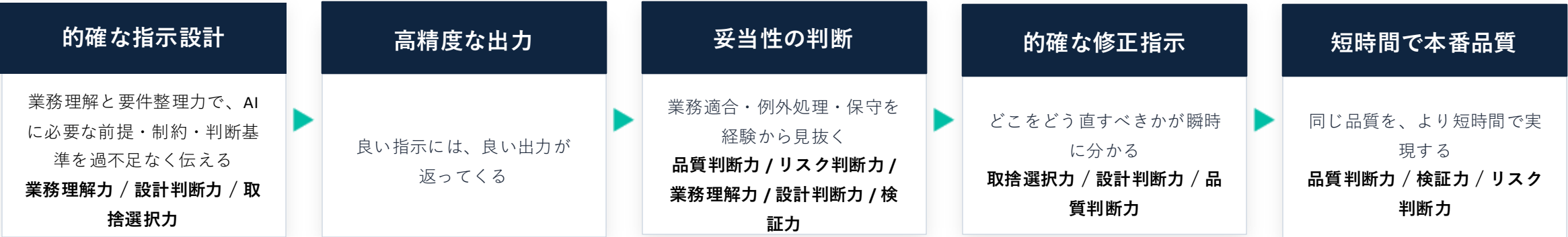
仕様・業務・テスト結果を照らし合わせ、問題点を早期に発見する

一方で、価値が下がるのは、判断を伴わない単純なコード作成・調査・形式的なテスト項目作成などです。

判断力 × AI で、生産性は跳ね上がる

AIの効果は、誰が使っても等しく現れるわけではありません。
的確な判断力・経験値が豊富な人材が使うとき、精度と速度が各ステップで積み上がり、生産性は単純な「ツール効果」を超えて跳ね上がります。

・ AI使用したソフトウェア開発の簡単なフロー



・ 的確な判断力・経験値の差は、AI利用コストにも現れる。

判断力が高い人 × AI	指示の精度 前提・制約が明確	AIの迷い 少ない	試行錯誤 1〜2回で着地	AI利用(トークン)コスト 低い
	指示の精度 前提・制約が曖昧	AIの迷い 多い	試行錯誤 繰り返し再生成	AI利用(トークン)コスト 高い

判断力なきAI活用は、コストとリスクを増幅させる

AI時代に判断力の欠如が招く問題は、単一ではない。
価値・品質・コスト・セキュリティなど複数の面で、同時に不利になる。

RISK 01

成果不一致

作ったものが、求められた価値に届かない

AIを使えば、短時間で形にすることはできますが、業務目的や利用者の意図を理解しないまま進めると、見た目は整っていても、現場で使えない成果物になります。
判断力がなければ、作業は速くなくても、本当に必要な価値には届きません。

RISK 02

品質リスク

正しくないものを正しそうに作る

AIの出力は、表面的には正しく見えます。
業務適合性・例外処理・保守性などを見抜けなければ、見えないリスクが積み上がっていきます。

RISK 03

コスト増大

トークン使用量が膨らみ、コストが上がる

曖昧な指示は、AI出力の精度を下げ、何度も再生成を繰り返すことになっていきます。
結果としてトークン消費が増え、AI利用コストが膨らみます。判断力の欠如は時間だけでなく、お金の面でも非効率を生みます。

RISK 04

セキュリティ事故

AIエージェントの誤操作で重大インシデント

重要ファイルの誤削除、顧客情報やライセンスキーの公開リポジトリへの流出は実際に起きている事故です。
AIエージェントに任せる範囲を誤ると、重大な事故につながります。
AIに与える権限や、任せる範囲を適切に判断する事が重要です。

マンパワー開発の終焉

これまで、作業量が増えれば人を増やすことが最も現実的な解決策でした。
しかしAI時代、人数を増やすメリットは大きく薄れ、デメリットの方が目立ちやすくなります。
さらに判断力なきAI利用者を増やせば、新たなリスクまで上乘せされます。

人数を増やすメリット

AIによって、優位性が薄れる

- ✕ 作業量を分散できる
→ AIで作業を圧縮 → 分散の必要性が低下
- ✕ 短期的な処理能力を上げられる
→ AIで処理能力を桁違いに引き上げる
- ✕ 複数工程を並列に進められる
→ 少人数でも複数工程を平行して進める事が可能

人数を増やすデメリット

人数を増やす事により発生する問題

人を増やすこと自体のデメリット

- 教育・レビューコストが増える
- 認識違い・手戻りが発生しやすい
- 品質や判断基準が人によってばらつく
- 管理・調整コストが膨らむ

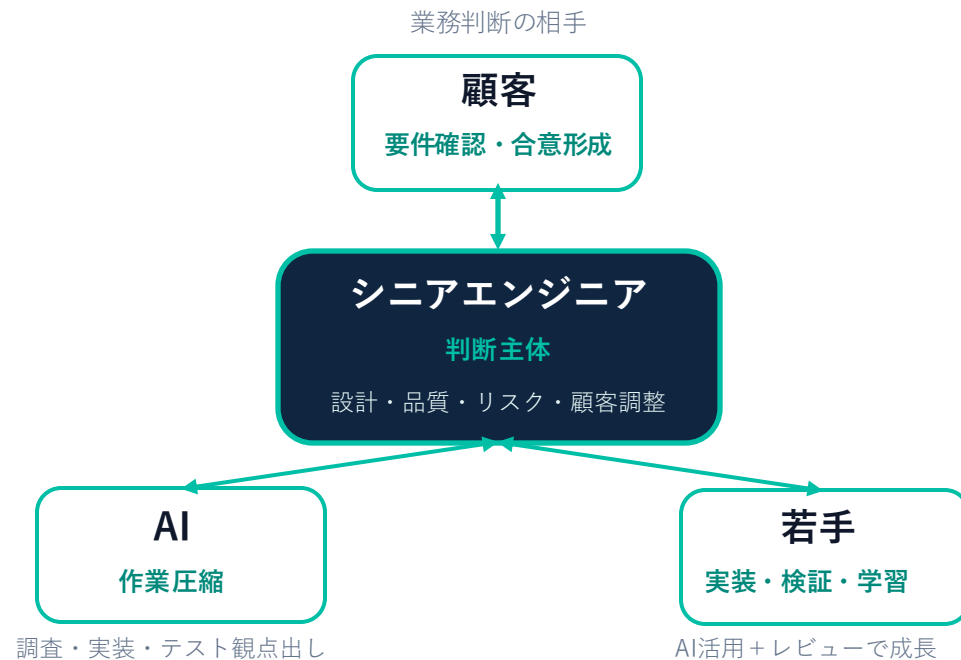
AI併用で新たに生じるデメリット

- 判断力なきAI利用で誤った成果物が量産される
- 曖昧な指示でAI再生成コストが膨らむ
- AIエージェント誤操作で重大なセキュリティ事故

これからは「人数で補う開発」から、「判断力とAIで圧縮する開発」へ

AI時代の開発体制モデル

人数を増やすだけでは、教育・レビュー・調整コストが膨らみます。
さらにAI活用には、出力を判断し、利用範囲を統制する力が必要です。
AI時代に必要なのは、判断主体を明確にした開発体制です。



シニアエンジニア主体の体制

顧客・若手・AIの間にシニアを判断主体として置くことで、AIの作業圧縮効果を活かしながら、人手依存の課題とAI活用リスクをまとめて解消します。

教育・レビューコストの増大

判断主体をシニアに絞り、レビュー観点と育成ポイントを明確にする。

認識違い・手戻り

顧客・若手・AIの間に判断主体を置き、業務理解と判断基準をそろえる。

品質・判断基準のばらつき

経験あるエンジニアの基準で、AI出力や成果物の妥当性を見極める。

管理・調整コストの増大

少人数で意思決定し、調整にかかる負荷を抑える。

AI利用リスク

AIに任せる範囲・指示内容・操作権限を明確にし、誤成果物・再生成コスト・誤操作を防ぐ。

結論： 作業はAIで圧縮し、判断は経験ある人材が担う。少人数でも、品質とスピードを両立できる体制へ。

若手育成も、「判断」を軸に再設計する

AI時代の若手育成は、単に作業をこなす経験から、判断のプロセスに早期から触れる経験へとシフトします。

これまで若手が経験を積む入口になっていた作業の多くは、AIで圧縮されていきます。だからこそ、育成の設計自体を見直す必要があります。

POINT 01

役割のシフト

「作業をこなす人」から「判断を学ぶ人」へ

単純作業を大量にこなして経験を積むだけの育成は、効率が下がっていきます。

これからの若手は、経験者の判断プロセスに早期から同席し、「なぜその設計にするのか」「なぜその修正が必要なのか」を学ぶ立場へ移ります。

POINT 02

AI は育成の加速装置

ただし、批評する訓練が必須

AIにより、複数の解法を比較し、疑問を即座に検証できるようになります。

ただし、出力をそのまま受け入れる癖がつくと、判断力は育ちません。AI出力を批評し、修正する訓練が重要です。

POINT 03

三者体制で育てる

シニア × AI × 若手

若手は、シニアの判断ループに参加しながら、AIを使って実装・検証を進めます。

設計レビュー・障害対応・顧客折衝にも早期から同席し、「コードが書ける」ではなく「判断できる」をゴールに置きます。

判断密度の高い経験を早期に積むことで、シニア化までの期間を短縮する。



第 3 部

私たちが提供できる価値

以上、AI時代で価値が高まる領域、人材について説明してきました。
ここからは、私たちがどのように顧客価値を生み出すのかお話しします

お客様にとっての変化

「完成まで分らない」から「早期に確かめながら進める」へ

AIと判断力を組み合わせることで、開発の進め方は「完成後に確認する」形から、「早い段階で確かめながら進める」形へ変わります。
画面・帳票・業務フローを早期に具体化することで、認識違いや業務とのズレを減らし、手戻りのリスクを抑えられます。

01

完成イメージが早く見える

画面・帳票・業務フローを早期に具体化

完成後まで見えにくかった使い勝手や業務の流れを、早い段階で確認できます。

「思っていたものと違う」を減らし、認識違いを早期に調整できます。

認識違いを最小化

02

関係者の合意を取りやすい

具体物を見ながら判断できる

資料だけで議論するのではなく、実際に動く画面や帳票を見ながら判断できます。

利用部門・管理者・開発側の認識を揃えやすくなります。

合意形成を早める

03

失敗リスクを下げられる

後工程での大きな手戻りを防ぐ

要件の認識違い、操作性の問題、業務フローの抜け漏れを開発前半で発見できます。
完成後に大きく作り直すリスクを抑えられます。

手戻りを抑える

早期に確認しながら進めることで、完成後のズレと手戻りを抑える。次スライドで少し詳細に説明。

早期プロトタイプで、顧客価値を具体化する

AI時代の提案では、初回ヒアリングで得た情報を、できるだけ早く具体案へ変換することが重要になります。

次回打ち合わせで画面案やプロトタイプを提示できれば、顧客は早い段階で認識違いに気づき、具体的な判断がしやすくなります。



初回コンタクトから動き出す事が重要です — ヒアリング内容をもとに、次回打ち合わせでは画面案やプロトタイプなど、具体的に確認できる形で提示します。

お客様の関与ポイント例

— 各STEPで何に関わり、何を受け取るか



AI活用で、自社製品開発を加速する

私たちは受託開発だけでなく、自社製品の開発にもAIを活用しています。

企画・設計・実装・改善の各工程でAIを使うことで、少人数でもスピード感を持って製品開発を進めています。

現在地 — すでに展開している製品

FlexGant

ガントチャートでの工程管理クラウド

建設・製造などの現場で、柔軟な工程管理を実現

DeepReport

自然言語で帳票出力

業務担当者が自然言語で直接指示。SQL自動生成 +
Excel・PDF出力

これから — AIを活用してさらに広げていく

業務ERP

販売管理・在庫管理・会計連携

受発注、在庫、請求、入金消込など、複雑な業務フローを支える基幹業務領域へ展開

現場業務アプリ

スマホ・タブレット × 業務サーバー連携

現場で発生する写真・作業記録・検査結果・進捗情報を、スマホやタブレットで入力し、業務サーバーと連携。現状回復・点検・検査・進捗管理などの現場業務へ展開。

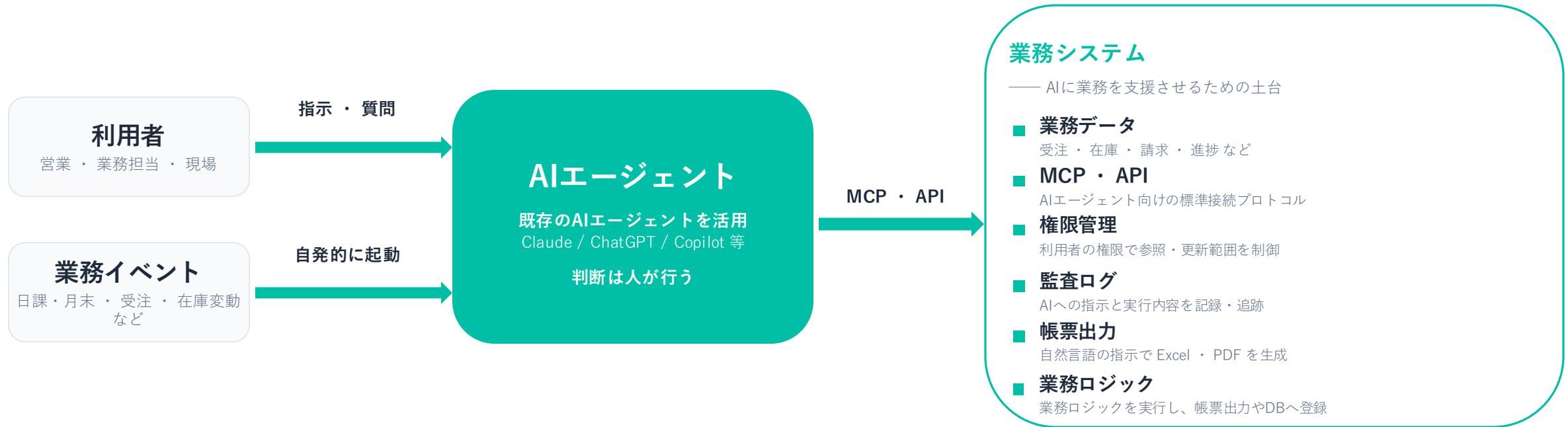
AIエージェント対応業務システム

既存AIエージェントと連携できるシステム

業務データ・API・帳票出力機能を整備し、AIエージェントが確認・入力補助・帳票作成を支援できる業務システムへ展開。

AIエージェントと連携する業務システムとは

AIエージェントを業務の入り口にし、業務システム側は画面ではなくAIから呼び出される機能・データ・統制を整備する



POINT 01

入口は AI、画面を全部作らない

業務担当の入口はAIエージェント（自然言語）。業務システム側は全画面を作らず、AIから呼び出される機能・データ・統制の整備に集中する。

POINT 02

AIエージェント本体は作らず、APIを作る

AI エージェントは Claude や ChatGPT など既存サービスを活用。私たちは、それを業務に組み込めるシステム側 (API) を設計する。

POINT 03

MCP で「AI に開く」設計

MCP は AI エージェント向けの標準接続プロトコル。特定 AI への依存を避け、将来の選択肢を残せる。

AIエージェントとの連携は、業務システムの新しい提供価値を広げる取り組みです。ここまで見てきた変化を踏まえ、最後に私たちが大切にすべき軸を整理します。

過去から、現在。そして未来へ。

ここまで、AIによって変わる開発の前提と、私たちが提供できる価値を整理してきました。
しかし、AI時代の正解は固定されたものではありません。
変わり続ける時代の中で、私たちは何を軸に進むべきかを考えます。

私たちは判断の時代にいる

AIによって、開発の前提は大きく変わりつつあります。
作業は圧縮され、判断力の重要性は高まり、
提案や製品開発の進め方も変わっていきます。
判断できる人材こそが、開発の中心に立っています。

しかし、

未来の正解は変わり続ける

重要なのは、学び続けること

AIがどこまで進化するかは、誰にも分かりません。
今日の正解が、明日も正解とは限りません。
だからこそ、変化を前提に学び、判断し、関係を築き続けることが必要です。
それはAI時代に限らず、私たちがこれまでも大切にしてきた姿勢です。

経費精算ワークフロー——経験不足者と経験者の違い

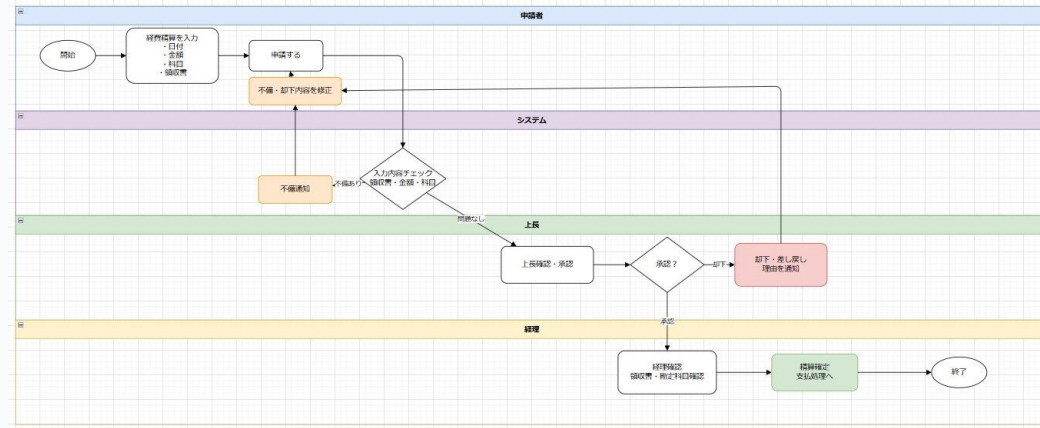
同じ「経費精算のワークフローを作って」という依頼でも、指示の解像度と出力の完成度に大きな差が生まれます。

経験不足者

依頼を字面どおりに受け取って実装

AIへの指示

「経費精算のワークフローを作って」



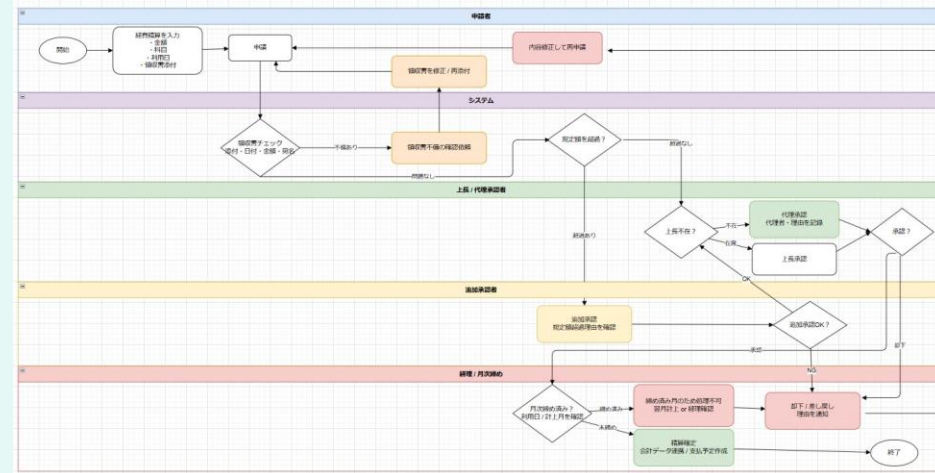
経験者

業務理解と例外処理を見越して指示を組み立てる

AIへの指示

「経費精算ワークフローを作って。ただし以下を考慮：

- ① 規定額超過時は追加承認
- ② 上長不在時の代理承認
- ③ 却下時の差し戻し
- ④ 領収書不備時の確認
- ⑤ 月次締めとの整合」



経験者は「実業務で何が起きるか」を先回りして指示に折り込み、AIから実用的な設計を引き出す。

経費精算ER図 —— 経験不足者と経験者の違い

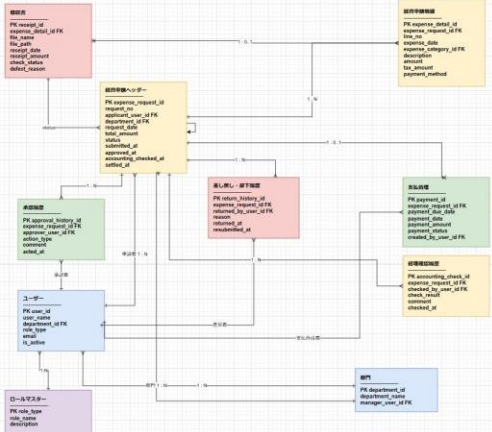
同じ「経費精算のER図を作って」という依頼でも、業務理解の深さが、データ構造の完成度に大きな差を生みます。

経験不足者

依頼を字面どおりに受け取って実装

AIへの指示

「経費精算のER図を作って」



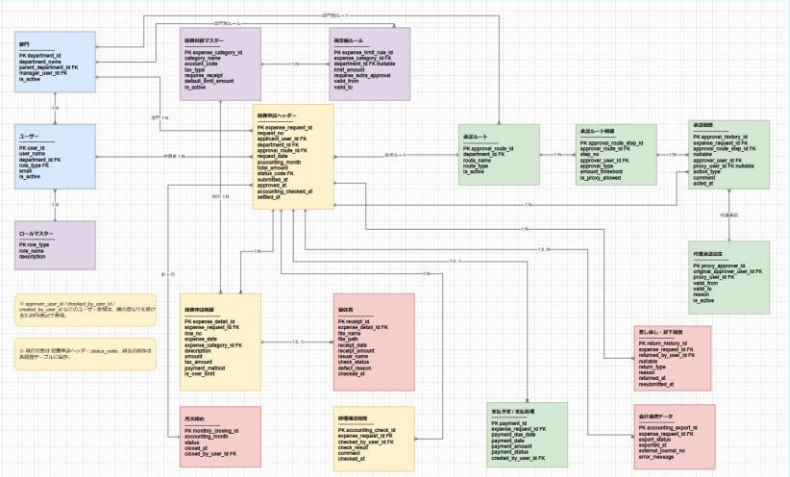
経験者

業務知識をもとに、必要な属性とリレーションを過不足なく設計

AIへの指示

「経費精算のER図を作って。ただし以下を考慮：

- ① 申請ステータスの状態管理
- ② 承認履歴・代理承認の追跡
- ③ 通貨・税区分・勘定科目
- ④ 領収書（添付ファイル）管理
- ⑤ 部署・規定額マスタとの関連



経験者は「業務に必要なデータの粒度と関連」を見越して指示に折り込み、AIから実用的な設計を引き出す。

コーディング —— 指示の解像度の差

同じ「この設計でコードを作って」という依頼でも、設計方針まで一緒に伝えるかどうかで、出力の構造に決定的な差が生まれる。

経験不足者

設計書をそのままAIに投げる

AIへの指示

「以下の基本設計をもとに、
経費精算の承認画面とAPIを
作成してください。」

設計書 + 一行の依頼。

設計書をAIに渡して、依頼は一行のみ。
「設計通り動けばOK」という発想で、
保守性・性能・監査などの非機能要件は
AIまかせ。指示の解像度はここで止まる。

経験者

設計書 + 守るべき設計方針を併せて指示する

AIへの指示

「以下の基本設計をもとに、経費精算の
承認画面とAPIを作成してください。
ただし、実装時は以下の設計方針を
必ず守ってください。
① 階層化 (Controller/Service/Domain/Repo)
② DTO設計 (Request/Response分離)
③ 共通化 (承認・差戻し・却下の重複排除)
④ 監査ログ (操作履歴の記録)
⑤ トランザクション (更新・履歴・ログ一括)
⑥ 一覧・検索性能 (N+1回避・ページング)」

設計書に加えて、守るべき設計方針を言語化する。

設計方針まで併せて指示する。
経験者は、画面やAPIの内容だけでなく、
階層化・DTO設計・共通化・監査ログ・トランザクション・性能など、
実装後の保守性や品質に関わる観点を先回りして伝える。
この差が、成果物のファイル構造やチェックリストに直接表れる。

経験者は「実装で何が起きるか」を見越して設計方針を言語化し、AIから保守可能な設計を引き出す。

コーディング —— 成果物の差

指示の解像度の差は、出力されたコードの構造に直接現れる。同じ機能でも、「形」がまったく違う。

経験不足者	設計書のまま投げた結果
ファイル構成 src/ └─ ApprovalController.java (約 500行) ロジック・DBアクセス・例外処理が 1ファイルに混在	設計方針への準拠 ✗ 階層化 ✗ 共通化 ✗ トランザクション ✗ DTO設計 ✗ 監査ログ ✗ 性能 (N+1回避)
経験者	設計方針を併せて指示した結果
ファイル構成 src/ ├─ controller/ → ApprovalController.java ├─ service/ → ApprovalService.java ├─ domain/ → ApprovalPolicy.java ├─ repository/ → ApprovalRepository.java └─ dto/ ├─ request/ (Approve / Return / Reject) └─ response/ (List / Detail)	設計方針への準拠 ✓ 階層化 ✓ 共通化 ✓ トランザクション ✓ DTO設計 ✓ 監査ログ ✓ 性能 (N+1回避)

指示の6項目が、そのままコードの構造として現れる —— これが業務理解と設計判断力の差。