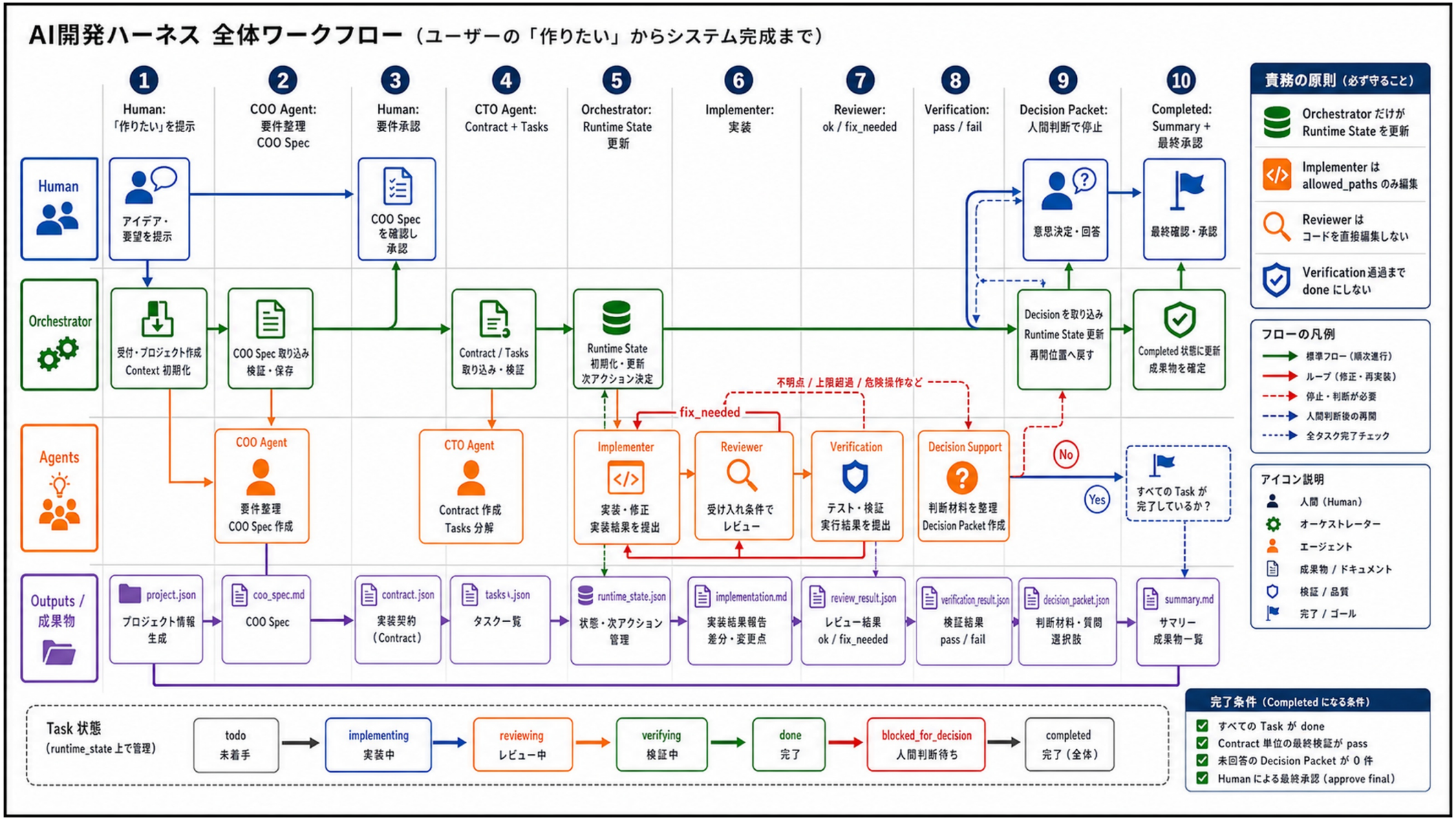
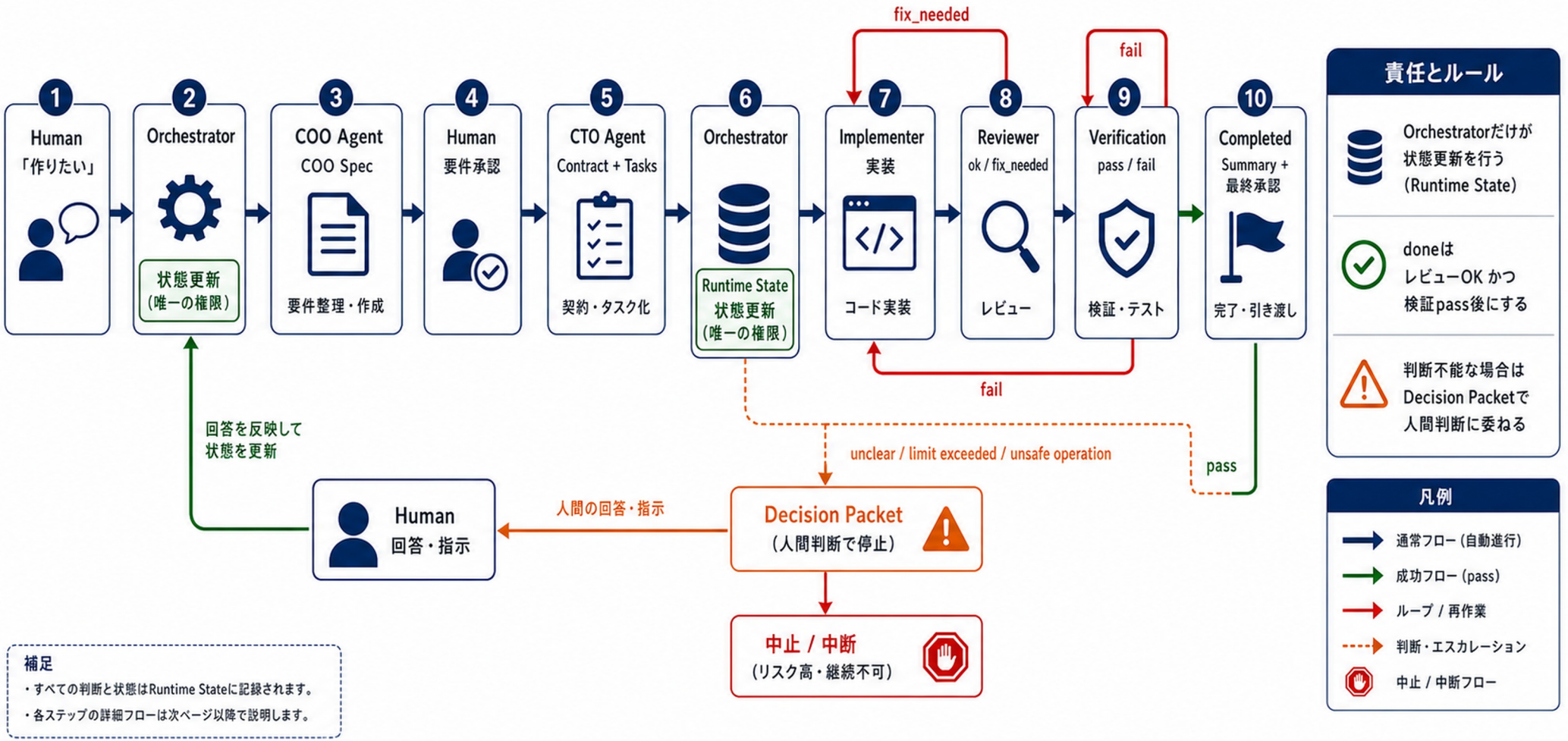
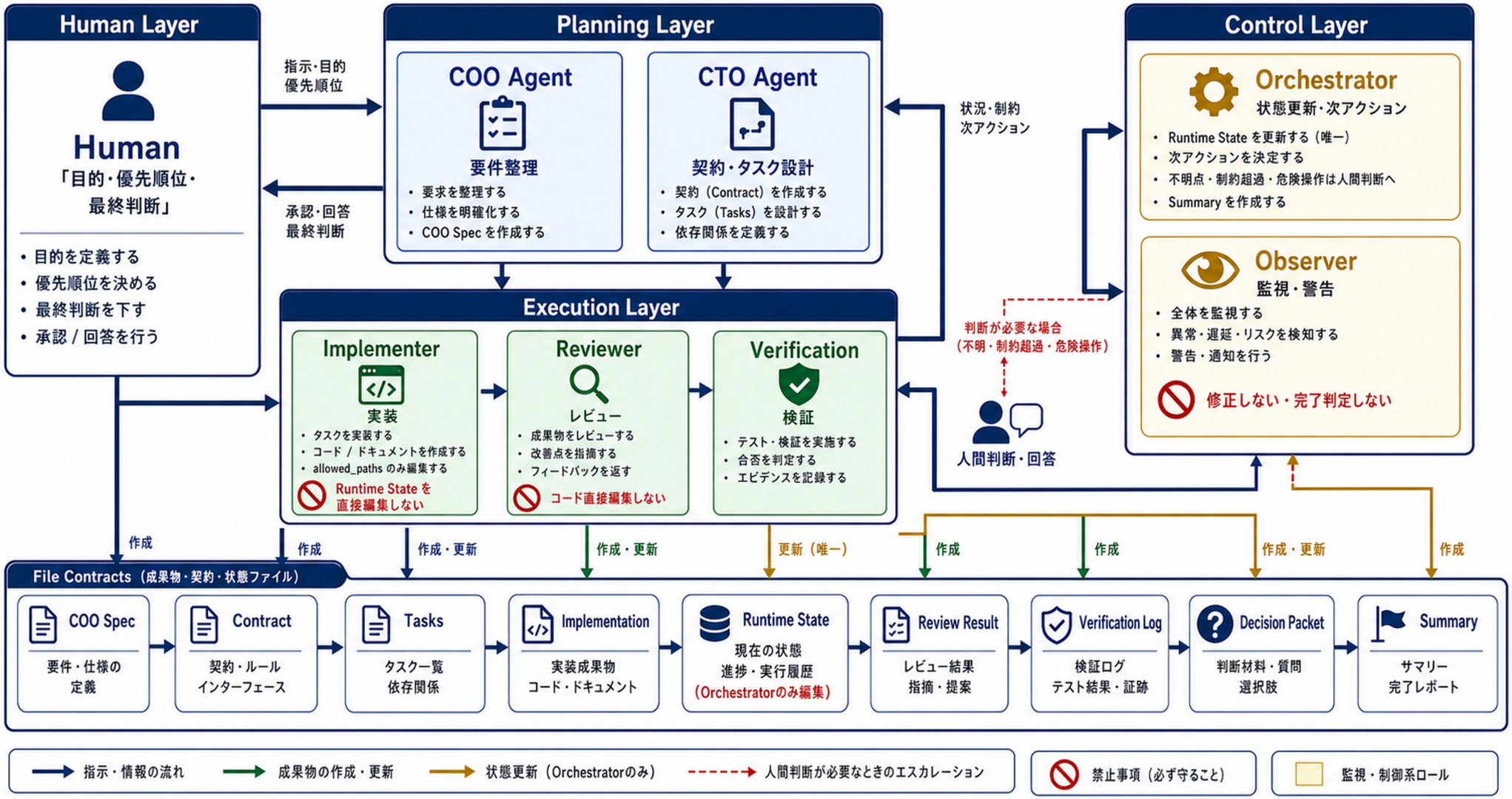

AI開発ハーネス 全体ワークフロー（ユーザーの「作りたい」からシステム完成まで）
1 Human: 「作りたい」を提示
2 COO Agent: 要件整理 COO Spec
3 Human: 要件承認
4 CTO Agent: Contract + Tasks
5 Orchestrator: Runtime State 更新
6 Implementer: 実装
7 Reviewer: ok / fix_needed
8 Verification: pass / fail
9 Decision Packet: 人間判断で停止
10 Completed: Summary + 最終承認
Human
Orchestrator
Agents
Outputs / 成果物
アイデア・要望を提示
COO Spec を確認し承認
意思決定・回答
最終確認・承認
受付・プロジェクト作成 Context 初期化
COO Spec 取り込み 検証・保存
Contract / Tasks 取り込み・検証
Runtime State 初期化・更新 次アクション決定
Decision を取り込み Runtime State 更新 再開位置へ戻す
Completed 状態に更新 成果物を確定
不明点 / 上限超過 / 危険操作など
fix_needed
COO Agent
要件整理 COO Spec 作成
CTO Agent
Contract 作成 Tasks 分解
Implementer
実装・修正 実装結果を提出
Reviewer
受け入れ条件で レビュー
Verification
テスト・検証 実行結果を提出
Decision Support
判断材料を整理 Decision Packet 作成
No
Yes
すべての Task が 完了しているか？
project.json プロジェクト情報 生成
coo_spec.md COO Spec
contract.json 実装契約 (Contract)
tasks.json タスク一覧
runtime_state.json 状態・次アクション 管理
implementation.md 実装結果報告 差分・変更点
review_result.json レビュー結果 ok / fix_needed
verification_result.json 検証結果 pass / fail
decision_packet.json 判断材料・質問 選択肢
summary.md サマリー 成果物一覧
責務の原則（必ず守ること）
Orchestrator だけが Runtime State を更新
Implementer は allowed_paths のみ編集
Reviewer は コードを直接編集しない
Verification 通過まで done にしない
フローの凡例
標準フロー（順次進行）
ループ（修正・再実装）
停止・判断が必要
人間判断後の再開
全タスク完了チェック
アイコン説明
人間（Human）
オーケストレーター
エージェント
成果物 / ドキュメント
検証 / 品質
完了 / ゴール
Task 状態 (runtime_state 上で管理)
todo 未着手
implementing 実装中
reviewing レビュー中
verifying 検証中
done 完了
blocked_for_decision 人間判断待ち
completed 完了（全体）
完了条件（Completed になる条件）
すべての Task が done
Contract 単位の最終検証が pass
未回答の Decision Packet が 0 件
Human による最終承認（approve final）

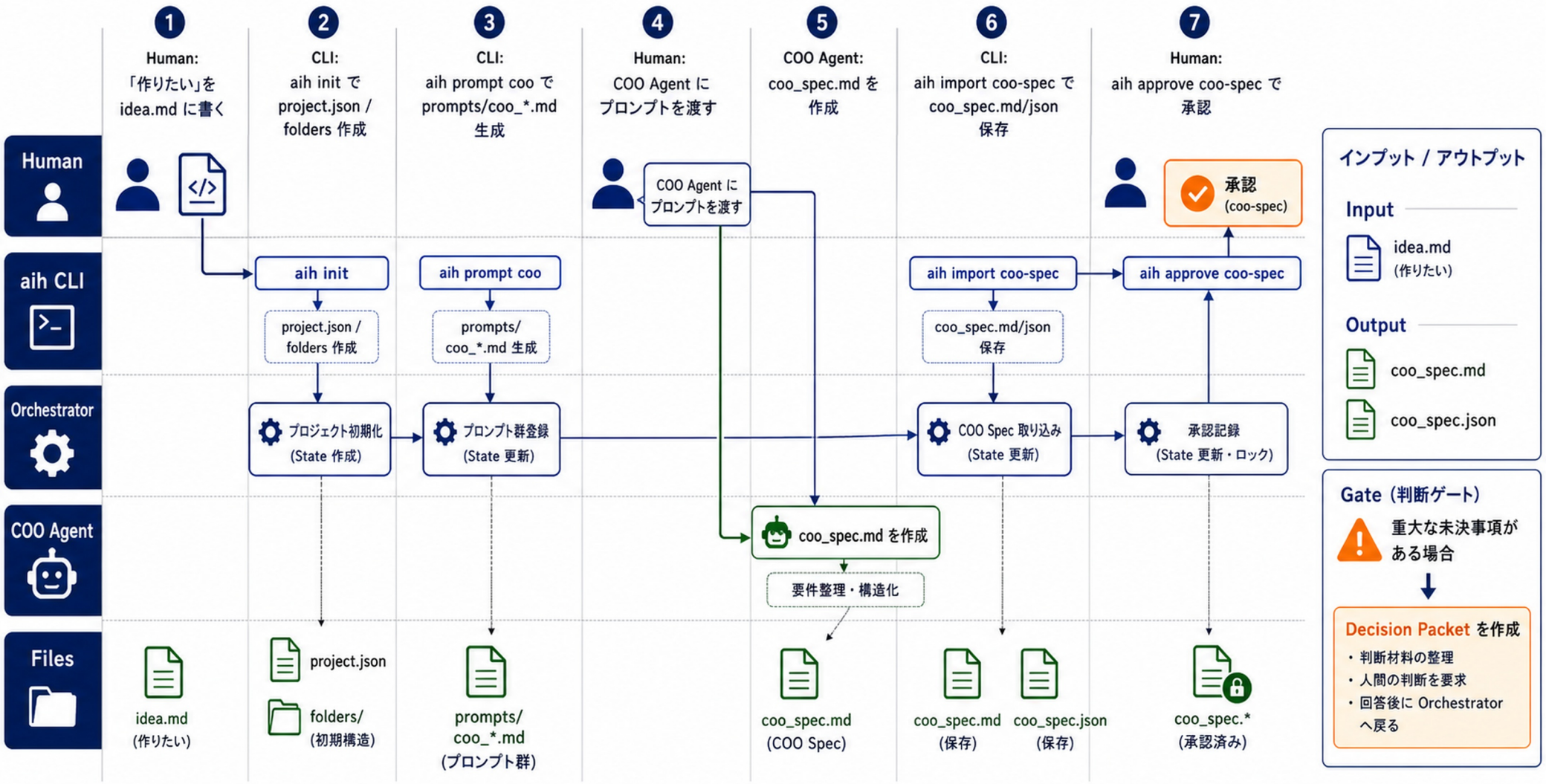
1. 全体ワークフロー（AI開発ハーネス）



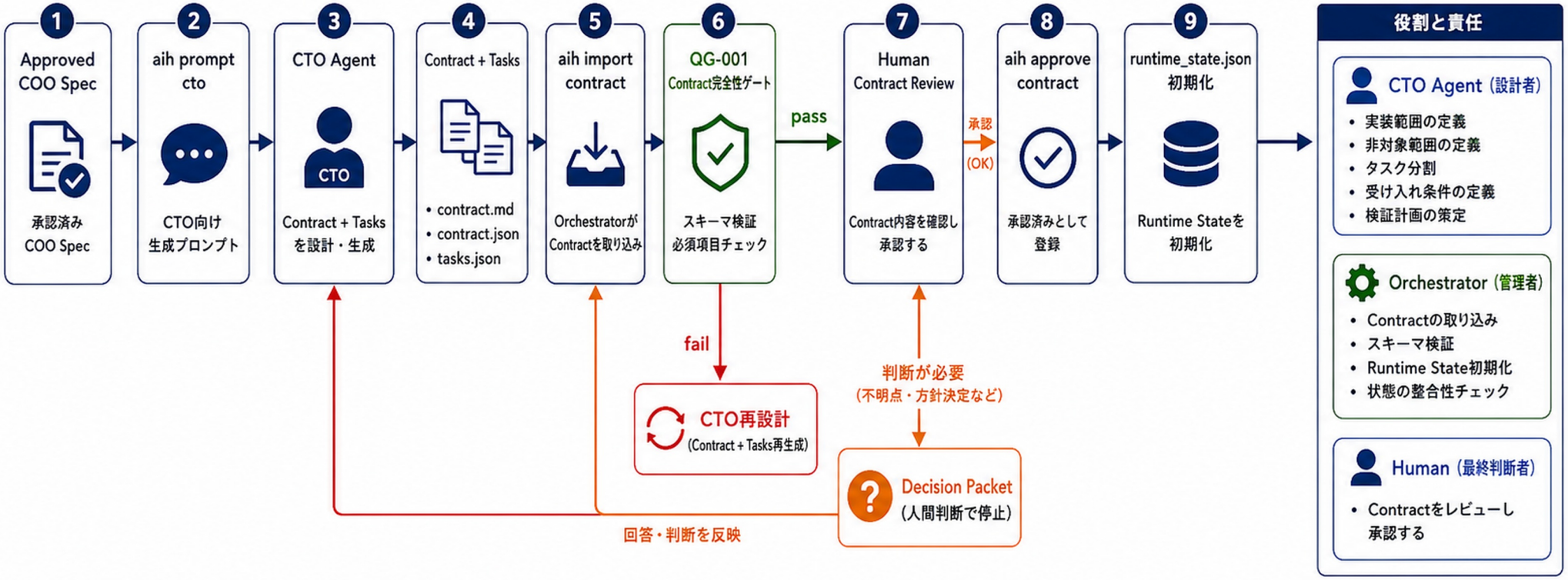
2. ロール責務マップ


Human Layer
Human
「目的・優先順位・最終判断」
• 目的を定義する
• 優先順位を決める
• 最終判断を下す
• 承認 / 回答を行う
指示・目的 優先順位
承認・回答 最終判断
Planning Layer
COO Agent
要件整理
• 要求を整理する
• 仕様を明確化する
• COO Spec を作成する
CTO Agent
契約・タスク設計
• 契約（Contract）を作成する
• タスク（Tasks）を設計する
• 依存関係を定義する
状況・制約 次アクション
Control Layer
Orchestrator
状態更新・次アクション
• Runtime State を更新する（唯一）
• 次アクションを決定する
• 不明点・制約超過・危険操作は人間判断へ
• Summary を作成する
Observer
監視・警告
• 全体を監視する
• 異常・遅延・リスクを検知する
• 警告・通知を行う
修正しない・完了判定しない
判断が必要な場合 （不明・制約超過・危険操作）
人間判断・回答
Execution Layer
Implementer
実装
• タスクを実装する
• コード / ドキュメントを作成する
• allowed_paths のみ編集する
Runtime State を 直接編集しない
Reviewer
レビュー
• 成果物をレビューする
• 改善点を指摘する
• フィードバックを返す
コード直接編集しない
Verification
検証
• テスト・検証を実施する
• 合否を判定する
• エビデンスを記録する
作成
作成
作成・更新
作成・更新
更新（唯一）
作成
作成
作成・更新
作成
File Contracts（成果物・契約・状態ファイル）
COO Spec
要件・仕様の 定義
Contract
契約・ルール インターフェース
Tasks
タスク一覧 依存関係
Implementation
実装成果物 コード・ドキュメント
Runtime State
現在の状態 進捗・実行履歴 (Orchestratorのみ編集)
Review Result
レビュー結果 指摘・提案
Verification Log
検証ログ テスト結果・証跡
Decision Packet
判断材料・質問 選択肢
Summary
サマリー 完了レポート
指示・情報の流れ
成果物の作成・更新
状態更新（Orchestratorのみ）
人間判断が必要なときのエスカレーション
禁止事項（必ず守ること）
監視・制御系ロール

3. Intake工程（要件取り込みフェーズ）



4. Planning工程（計画フェーズ）



入力（Inputs）

coo_spec.md / coo_spec.json
（承認済みCOO Spec）

出力（Outputs）

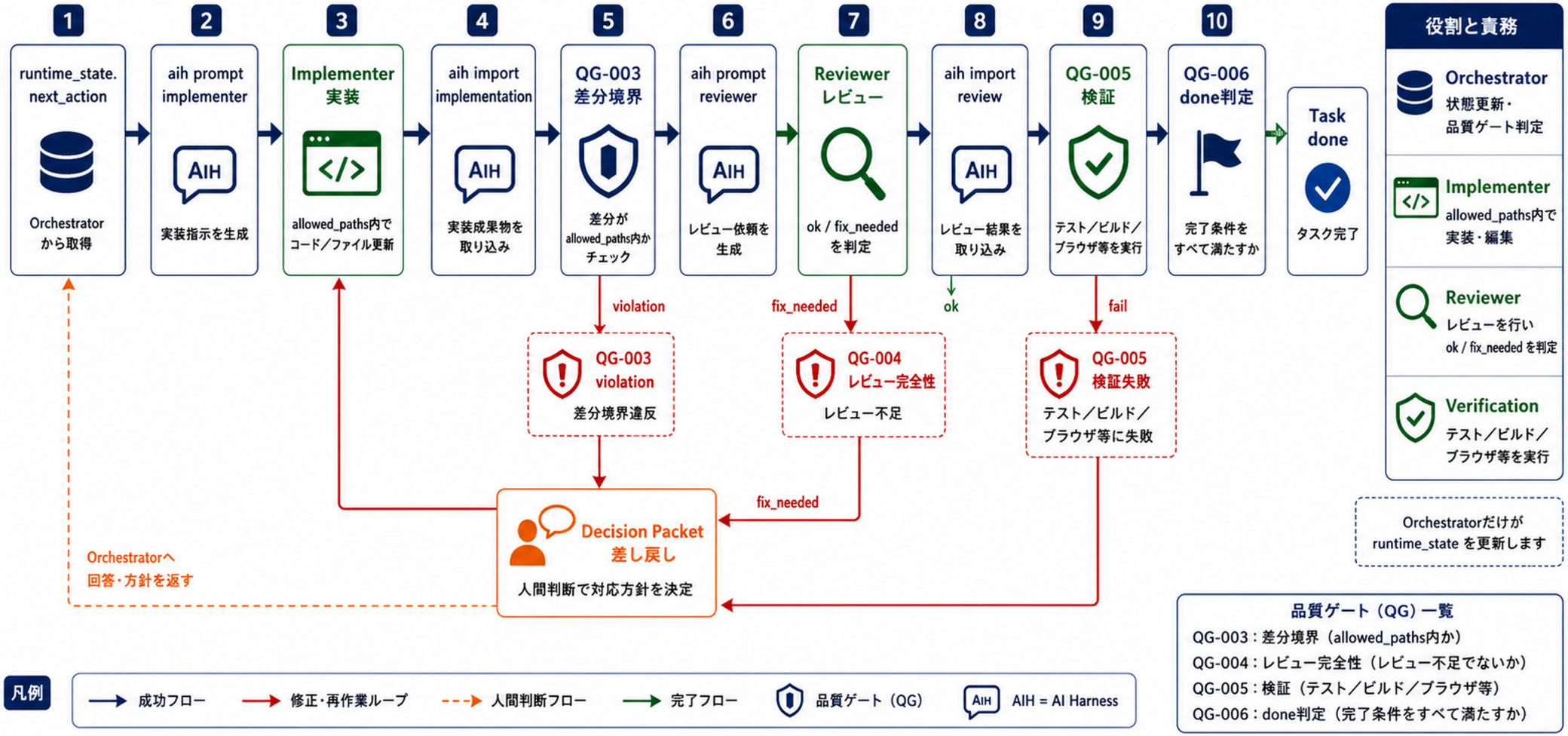
- contract.md（契約書・人間可読）
- contract.json（構造化データ）
- tasks.json（タスク一覧）
- runtime_state.json（初期状態）

ポイント

- ContractはAI開発の「憲法」です。
- 完全性ゲートを通過し、人間が承認してから実行フェーズへ進みます。
- 不明点や方針判断が必要な場合は、Decision Packetで停止します。

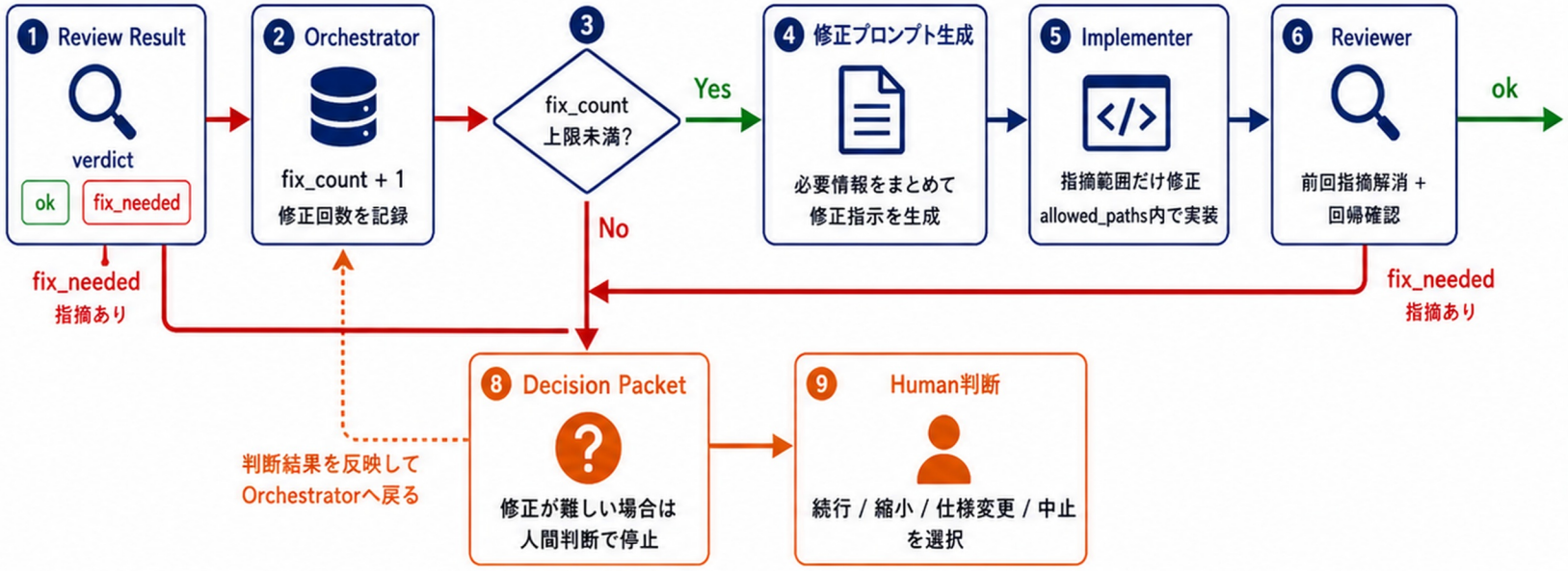
5. Execution工程

1タスクの実行フロー（AI開発ハーネス）



6. レビュー / 修正ループ

レビューで指摘があった場合の修正サイクル（fix_needed → ok まで）



役割と責務

Reviewer（レビューアー）
- 指摘・根拠・該当ACを明記
- コードは直接編集しない

Orchestrator（オーケストレータ）
- fix_countを管理・記録
- 上限判定を行う

Implementer（実装エージェント）
- 修正は allowed_paths 内で実施
- 指示範囲外は変更しない

Human（人間）
- 続行 / 縮小 / 仕様変更 / 中止を判断
- 判断内容で次のアクションを決定

修正プロンプトの必須内容

1 対象 Task	修正対象の Task ID / タイトルを明記
2 該当 Acceptance Criteria	該当するAC番号と内容を明記
3 指摘の根拠	Reviewerの指摘内容と検出根拠（エビデンス）
4 変更可能ファイル範囲	allowed_paths に含まれるファイル一覧・範囲
5 追加検証	修正後に実施すべき検証や確認ポイント

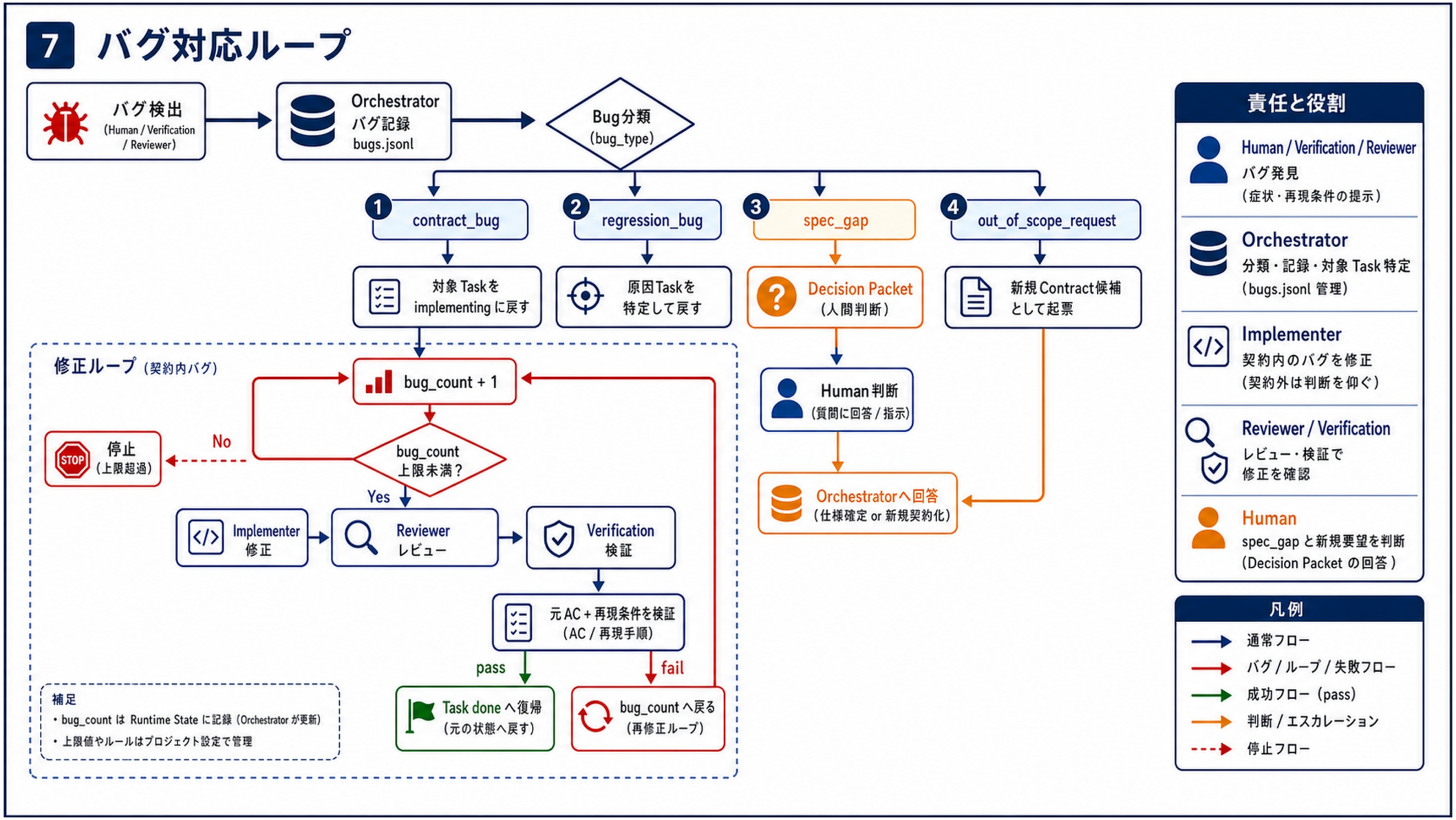
修正プロンプト（例）

1. Task-12 ユーザー登録APIの作成
2. AC-2, AC-3
3. 入力バリデーションが不足しており、空文字で登録できてしまう（テストケース TC-05 失敗）
4. allowed_paths/api/user/、allowed_paths/tests/test_user.py
5. 正常系・異常系のユニットテストを追加し、全ケースがパスすることを確認

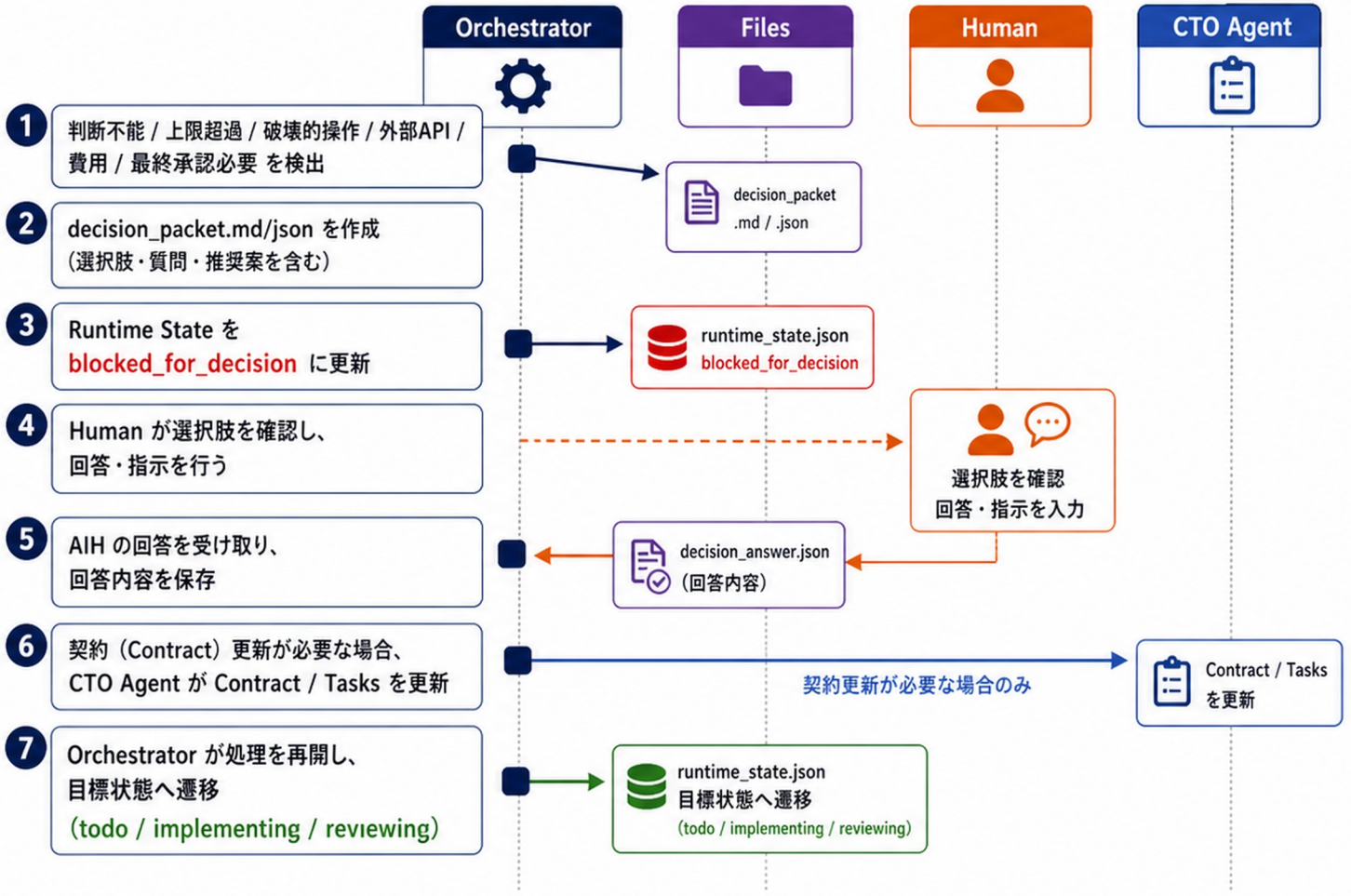
凡例

- 通常のフロー（順方向）
- 修正ループ（fix_needed）
- 人間判断フロー（上限超過・難易度高）
- レビューOK後のフロー

上限の考え方
- 例：最大修正回数 = 3回（プロジェクトで設定）
- 上限に到達した場合は、必ず 人間判断へ


7 バグ対応ループ
バグ検出
(Human / Verification / Reviewer)
Orchestrator
バグ記録
bugs.jsonl
Bug分類
(bug_type)
1 contract_bug
2 regression_bug
3 spec_gap
4 out_of_scope_request
対象 Taskを implementing に戻す
原因Taskを特定して戻す
Decision Packet
(人間判断)
新規Contract候補として起票
修正ループ (契約内バグ)
bug_count + 1
No
停止
(上限超過)
bug_count 上限未満？
Yes
Implementer 修正
Reviewer レビュー
Verification 検証
元AC + 再現条件を検証
(AC / 再現手順)
pass
fail
Task done へ復帰
(元の状態へ戻す)
bug_count へ戻る
(再修正ループ)
Human 判断
(質問に回答 / 指示)
Orchestratorへ回答
(仕様確定 or 新規契約化)
補足
・bug_count は Runtime State に記録 (Orchestrator が更新)
・上限値やルールはプロジェクト設定で管理
責任と役割
Human / Verification / Reviewer
バグ発見
(症状・再現条件の提示)
Orchestrator
分類・記録・対象 Task特定
(bugs.jsonl 管理)
Implementer
契約内のバグを修正
(契約外は判断を仰ぐ)
Reviewer / Verification
レビュー・検証で
修正を確認
Human
spec_gap と新規要望を判断
(Decision Packet の回答)
凡例
通常フロー
バグ / ループ / 失敗フロー
成功フロー (pass)
判断 / エスカレーション
停止フロー

8. Human Decision工程　人間判断が必要なときの処理フロー（AI開発ハーネス）



Decision Packet（必須フィールド）

判断理由	なぜ人間の判断が必要か
現在状況	Runtime State / Task 状況 / 直前の結果
選択肢（2〜3個）	選べるオプションと影響
推奨案	AI の推奨と理由
Human への質問	人間に確認したい事項
resume target	回答後の目標状態 (todo / implementing / reviewing)

責任とルール

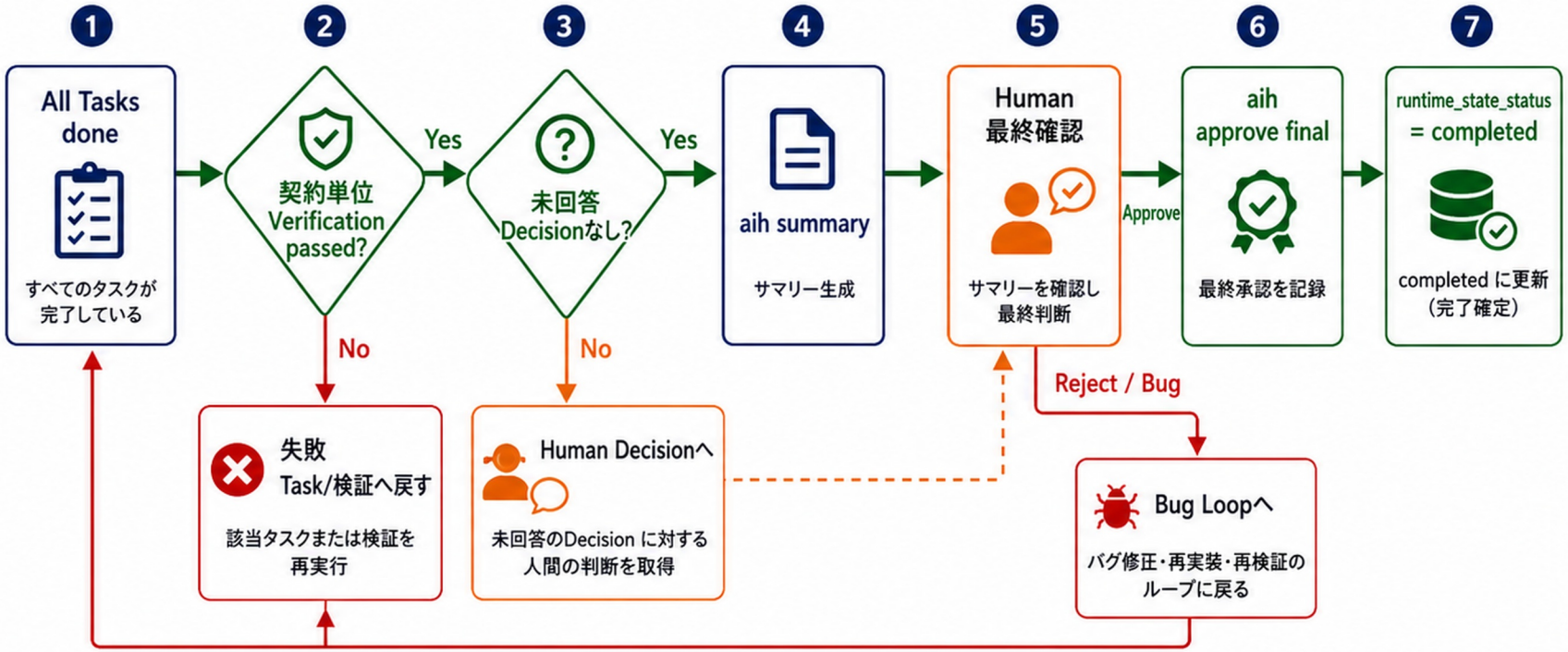
Orchestrator	**停止・記録・再開** 検出 → Decision Packet 作成 → blocked にする → 回答保存 → 再開
Human	**判断** 選択肢を確認し、最終判断を行う
CTO Agent	**契約更新が必要な時だけ** Contract / Tasks を更新する (不要な場合は更新しない)

システム処理（自動）　情報の提示・取得（人間とのやり取り）　契約更新の連携（必要な場合のみ）　再開・進行（自動）　停止状態（blocked）　ファイル生成・保存

09 9. Completion工程

AI開発ハーネス

すべてのタスクと検証が完了し、人間の最終承認を得て、システムを completed 状態にします。



サマリー（aih summary）に含まれる内容

- 作ったもの
- 主な変更点
- 満たした受け入れ条件
- 実行した検証
- 既知の制約 / 残課題
- 成果物の場所

ロールと責任

- **Orchestrator**
  完了条件の照合・summary生成・completed化（状態更新）
- **Verification**
  契約単位の最終検証を実施し、pass を確定
- **Human**
  サマリー確認・残終判断・承認

成功フロー（完了まで）
すべての条件を満たし、最終承認後に completed 状態へ

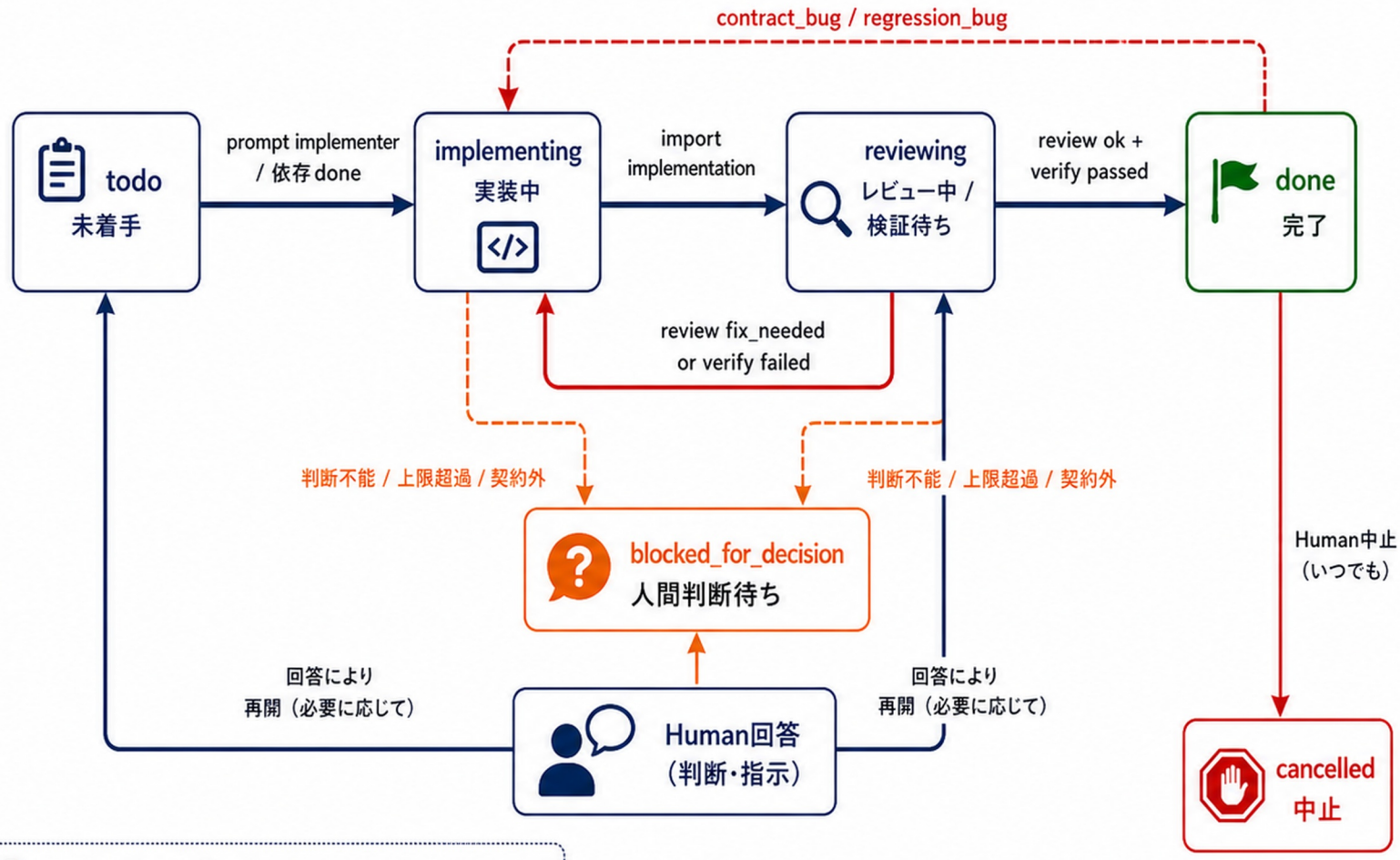
判断待ちフロー
未回答の Decision がある場合は Human Decision へ

失敗フロー（戻り）
検証失敗や Reject / Bug は Bug Loop へ戻り再実行

完了条件

- すべてのタスクが done
- 契約単位の Verification がすべて pass
- 未回答の Decision が存在しない
- Human の最終承認（approve final）を取得

10. Task状態遷移（AI開発ハーネス）



責務・ルール

- Orchestratorだけが Task状態を変更する（唯一の更新権限）
- Human回答なしで blockedから復帰しない
- 検証（verify）なしで doneにしない
- 契約バグ・リグレッションは doneからimplementingに戻して再実装する

凡例

- 通常の状態遷移
- バグ・リグレッション / 中止
- 判断・エスカレーション
- 完了状態

Taskが Doneになる条件

レビューOK かつ 検証pass（すべての必須チェックを通過）後のみ

11. Runtime Stateと次アクション判定

Orchestratorは runtime_state.json を読み取り、優先順位に従って next_action を決定する

単一の真実の源泉 (Source of Truth)

runtime_state.json

Orchestratorが読み取り
次アクションを決定

主要フィールド（例）

- **status**
  全体の状態（例：implementing）
- **current_task_id**
  現在処理中のタスクID（nullable）
- **tasks**
  各タスクの状態一覧
  - todo | implementing | reviewing
  - verified | done | blocked
- **review_count**
  現在のレビュー試行回数
- **fix_count**
  現在の修正試行回数
- **next_action**
  Orchestratorが決定した次の行動
- **pending_decision_packet**
  未回答の Decision Packet（nullable）

※ フィールド名は一例です。実装に合わせて拡張可能です。

Orchestrator：次アクション判定フロー（優先順位順）

1. 未回答の Decisionあり？
   - Yes → next_action = answer_decision
     人間の回答を待つ
   - No ↓
2. 状態不整合 あり？
   - Yes → next_action = doctor / decision
     状態修復 または 人間判断
   - No ↓
3. reviewing の Taskあり？
   - Yes → next_action = review or verify
     レビュー継続 または 検証へ
   - No ↓
4. implementing の Taskあり？
   - Yes → next_action = import_implementation
     実装成果物の取込待ち
   - No ↓
5. 実行可能な todo あり？
   - Yes → next_action = generate_implementer_prompt
     実装タスク生成 → Implementerへ指示
   - No ↓
6. 全Task が done？
   - Yes → next_action = summary / final_approval
     サマリー作成 → 最終承認を待つ
   - No ↓
7. それ以外（想定外の状態）
   - Yes → next_action = state_conflict
     状態競合：人間判断を要する

役割と責務

Orchestrator（唯一の更新者）

- Runtime Stateの唯一の更新者
- next_action を決定・記録
- 状態の整合性を管理

Observer（監視者）

- runtime_state.json を読み取り
- 状態を監視・警告のみ
- 状態を変更しない

Agents（各エージェント）

- 自分の担当成果物のみ出力
- allowed_paths の範囲で活動
- Runtime State を直接変更しない
  （Orchestrator が反映）

凡例

- 判定（状態チェック）
- Yes（条件を満たす）
- No（次の判定へ）
- 次アクション（next_action）

判定のポイント

1 → 2 → 3 の順に
優先度が高い判断を処理

状態不整合の例
- 参照不整合
- 必須フィールド欠落
- 矛盾する状態遷移 など

reviewing がある場合
→ まずはレビュー完了または
検証に進める

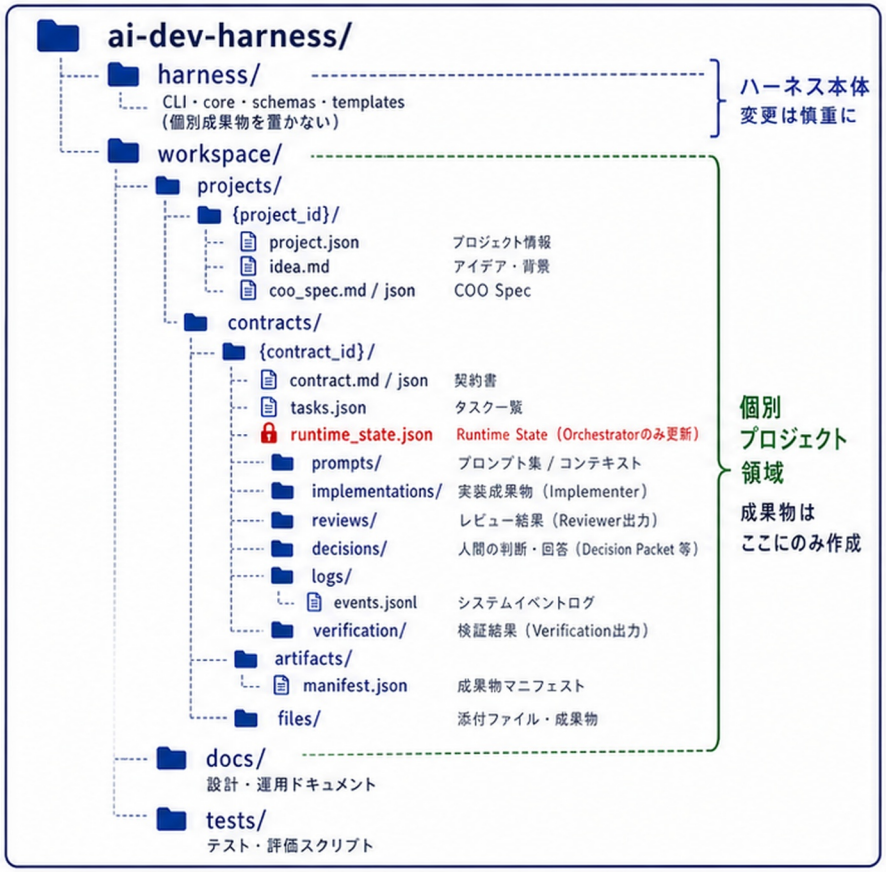
implementing がある場合
→ 実装の取込が完了するまで
待つ

todo がある場合
→ 次の実装に進むのが
最も有効

全Task doneで初めて
サマリーと最終承認に
進む

12 フォルダ責務分離

AI開発ハーネスのフォルダ構成と、各ロールのアクセス責務を定義します。



アクセス責務マトリクス

ロール	読み取り	作成	更新	削除	対象パス (主なもの)	備考
Orchestrator	○	○	○	○	runtime_state.json prompts/ decisions/ artifacts/manifest.json summary (完了レポート)	Runtime Stateは Orchestratorのみ 更新可能
Implementer	○	○	△ (条件付き)	×	implementations/ (allowed_paths のみ)	allowed_paths 外は 編集しない
Reviewer	○	○	×	×	reviews/	コードを直接 編集しない
Verification	○	○	○	×	logs/verification/	検証結果のみ 出力・更新
Human	○	○	○	△ (条件付き)	decisions/ (承認・回答ファイル)	最終承認・ 判断・回答

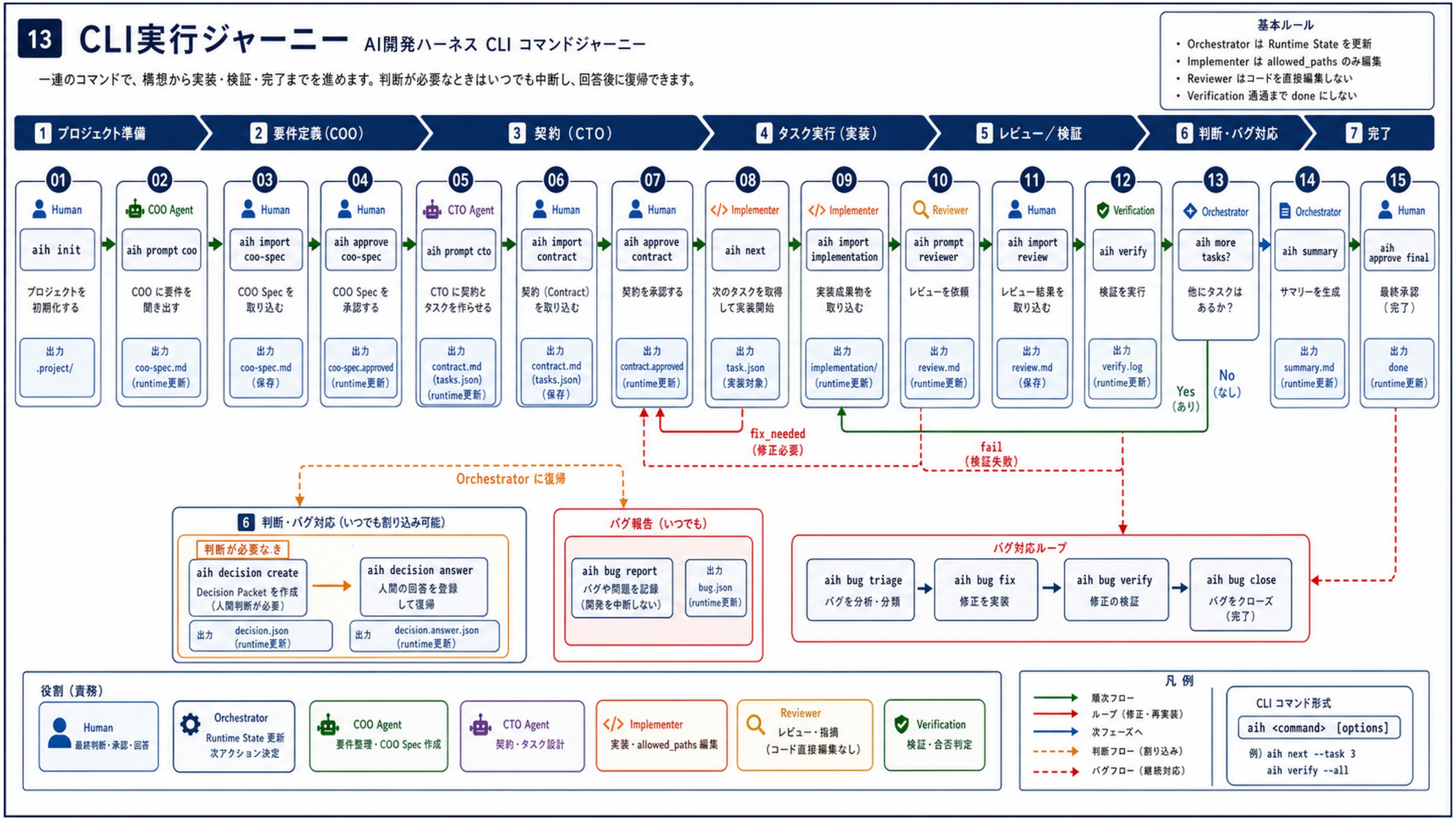
○ 許可 (推奨)　△ 条件付き　× 禁止 (しないこと)

ハーネス本体と個別プロジェクトを混ぜない。

ハーネス本体 (harness/) には個別プロジェクトの成果物を置かないこと。
すべての成果物は workspace/projects/{project_id}/contracts/{contract_id}/ 配下に保存する。

フォルダ　ファイル　制限付き (特定ロールのみ)　|　○ 許可 (推奨)　△ 条件付き　× 禁止

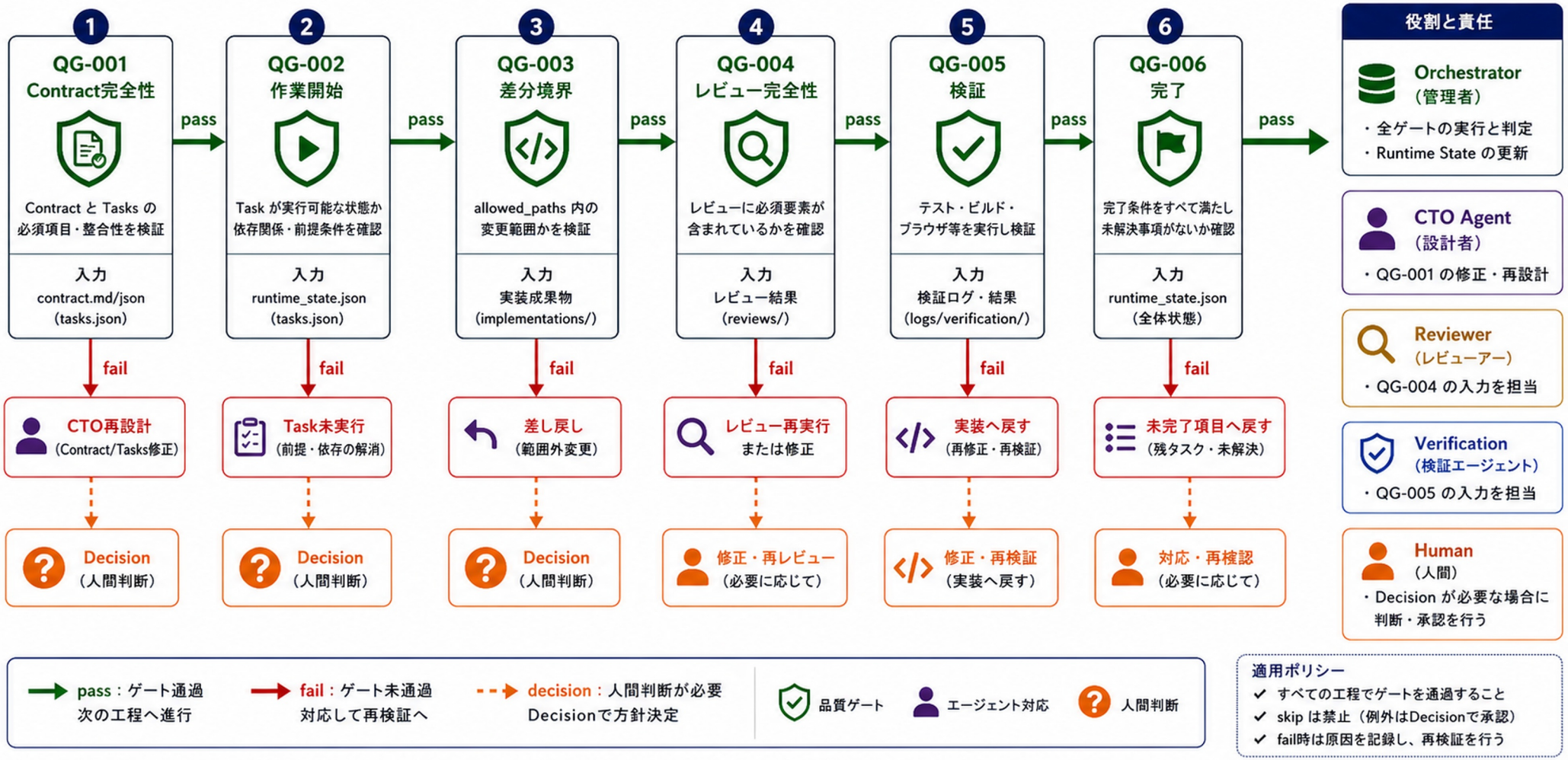
原則：最小権限の原則を適用し、各ロールは必要な範囲にのみアクセスする。
例外：人間判断が必要な場合は Decision Packet に出力し、Human の回答を待つ。


13 CLI実行ジャーニー AI開発ハーネス CLI コマンドジャーニー
一連のコマンドで、構想から実装・検証・完了までを進めます。判断が必要なときはいつでも中断し、回答後に復帰できます。
基本ルール
• Orchestrator は Runtime State を更新
• Implementer は allowed_paths のみ編集
• Reviewer はコードを直接編集しない
• Verification 通過まで done にしない
1 プロジェクト準備
2 要件定義（COO）
3 契約（CTO）
4 タスク実行（実装）
5 レビュー／検証
6 判断・バグ対応
7 完了
01 Human aih init プロジェクトを初期化する 出力 .project/
02 COO Agent aih prompt coo COO に要件を聞き出す 出力 coo-spec.md (runtime更新)
03 Human aih import coo-spec COO Spec を取り込む 出力 coo-spec.md (保存)
04 Human aih approve coo-spec COO Spec を承認する 出力 coo-spec.approved (runtime更新)
05 CTO Agent aih prompt cto CTO に契約とタスクを作らせる 出力 contract.md (tasks.json) (runtime更新)
06 Human aih import contract 契約（Contract）を取り込む 出力 contract.md (tasks.json) (保存)
07 Human aih approve contract 契約を承認する 出力 contract.approved (runtime更新)
08 Implementer aih next 次のタスクを取得して実装開始 出力 task.json (実装対象)
09 Implementer aih import implementation 実装成果物を取り込む 出力 implementation/ (runtime更新)
10 Reviewer aih prompt reviewer レビューを依頼 出力 review.md (runtime更新)
11 Human aih import review レビュー結果を取り込む 出力 review.md (保存)
12 Verification aih verify 検証を実行 出力 verify.log (runtime更新)
13 Orchestrator aih more tasks? 他にタスクはあるか?
Yes (あり)
No (なし)
14 Orchestrator aih summary サマリーを生成 出力 summary.md (runtime更新)
15 Human aih approve final 最終承認（完了） 出力 done (runtime更新)
fix_needed (修正必要)
fail (検証失敗)
Orchestrator に復帰
6 判断・バグ対応（いつでも割り込み可能）
判断が必要なき
aih decision create Decision Packet を作成（人間判断が必要） 出力 decision.json (runtime更新)
aih decision answer 人間の回答を登録して復帰 出力 decision.answer.json (runtime更新)
バグ報告（いつでも）
aih bug report バグや問題を記録（開発を中断しない） 出力 bug.json (runtime更新)
バグ対応ループ
aih bug triage バグを分析・分類
aih bug fix 修正を実装
aih bug verify 修正の検証
aih bug close バグをクローズ（完了）
役割（責務）
Human 最終判断・承認・回答
Orchestrator Runtime State 更新 次アクション決定
COO Agent 要件整理・COO Spec 作成
CTO Agent 契約・タスク設計
Implementer 実装・allowed_paths 編集
Reviewer レビュー・指摘（コード直接編集なし）
Verification 検証・合否判定
凡例
順次フロー
ループ（修正・再実装）
次フェーズへ
判断フロー（割り込み）
バグフロー（継続対応）
CLI コマンド形式
aih <command> [options]
例) aih next --task 3
aih verify --all

14. 品質ゲート

AI開発ハーネス 品質ゲートパイプライン

各フェーズの成果物と状態を検証し、品質を担保します。全ゲートを通過してはじめて次工程へ進みます。



15. 失敗モード対応（AI開発ハーネス）

失敗を早期に検知し、適切に分類・対応して安全にリカバリします。

失敗の検知元（例）

- **Human** — 違和感・異常を報告
- **Reviewer** — 仕様逸脱・レビュー品質低下
- **Verification** — 検証不足・検証失敗
- **Observer** — 不整合・異常な挙動を検知
- **Orchestrator** — ルール違反・上限超過

すべての対応の原則

- Orchestratorが分類し、停止または制限
- 原因の記録（events.jsonl / logs）を必ず残す
- 必要に応じてDecision Packet で人間判断へ
- 再開条件を満たすまで次へ進まない

緊急停止条件（即時停止）

STOP

- 破壊的操作の検知（FM-009）
- フォルダ汚染の疑い（FM-006）
- 重大なセキュリティリスクの検知

Failure detected 失敗を検知

失敗モード分類（Orchestrator）

FM-001 無限修正ループ

fix_needed が繰り返され、fix_count 上限に到達

対応：Decision Packet へ

FM-002 仕様逸脱（契約外作業）

実装・出力が Contract / Tasks / allowed_paths を逸脱

対応：差し戻し / Decision

FM-003 コンテキスト忘れ・脱線

指示や契約を忘れて脱線、目的から外れた出力

対応：Context 再注入

2回連続で再発 → Decision へ

FM-004 レビュー品質不足

レビューが表面的、AC未確認、根拠・エビデンス不足

対応：レビュー再実行

FM-005 検証不足・検証失敗

検証が不十分 / 検証結果が fail

対応：戻して検証やり直し

done 状態にしない

FM-006 フォルダ汚染・境界破壊

allowed_paths 外の編集や他契約・プロジェクトへの影響

対応：停止してログ化

調査後に Decision へ

FM-008 状態不整合

Runtime State と実際の成果物・状態が一致しない

対応：doctor 実行 / Decision

FM-009 破壊的操作の検知

DELETE / 強制上書き / 外部システムへの影響あり

対応：即時停止（最優先）

Human 判断が必要

FM-010 完了誤判定

検証・レビュー未完了なのに done と判定された

対応：レビューへ戻す

リカバリ後の再開

再開条件を満たしたら runtime_state.next_action に基づき通常フローへ復帰

ロールの責務

ロール	責任
Orchestrator（管理者）	・失敗の分類・記録・停止 ・Runtime State 更新 ・リカバリ条件の評価
Observer（監視者）	・ログ・状態を監視 ・異常パターンの検知と警告（修正は行わない）
Reviewer（レビューアー）	・仕様逸脱・低品質レビューを発見 ・根拠・AC 未確認を指摘
Verification（検証者）	・検証不足・検証失敗を発見 ・検証結果を記録
Human（人間）	・Decision Packet を確認 ・最終判断（続行 / 変更 / 中止）

Decision Packet（作成が必要な場合）

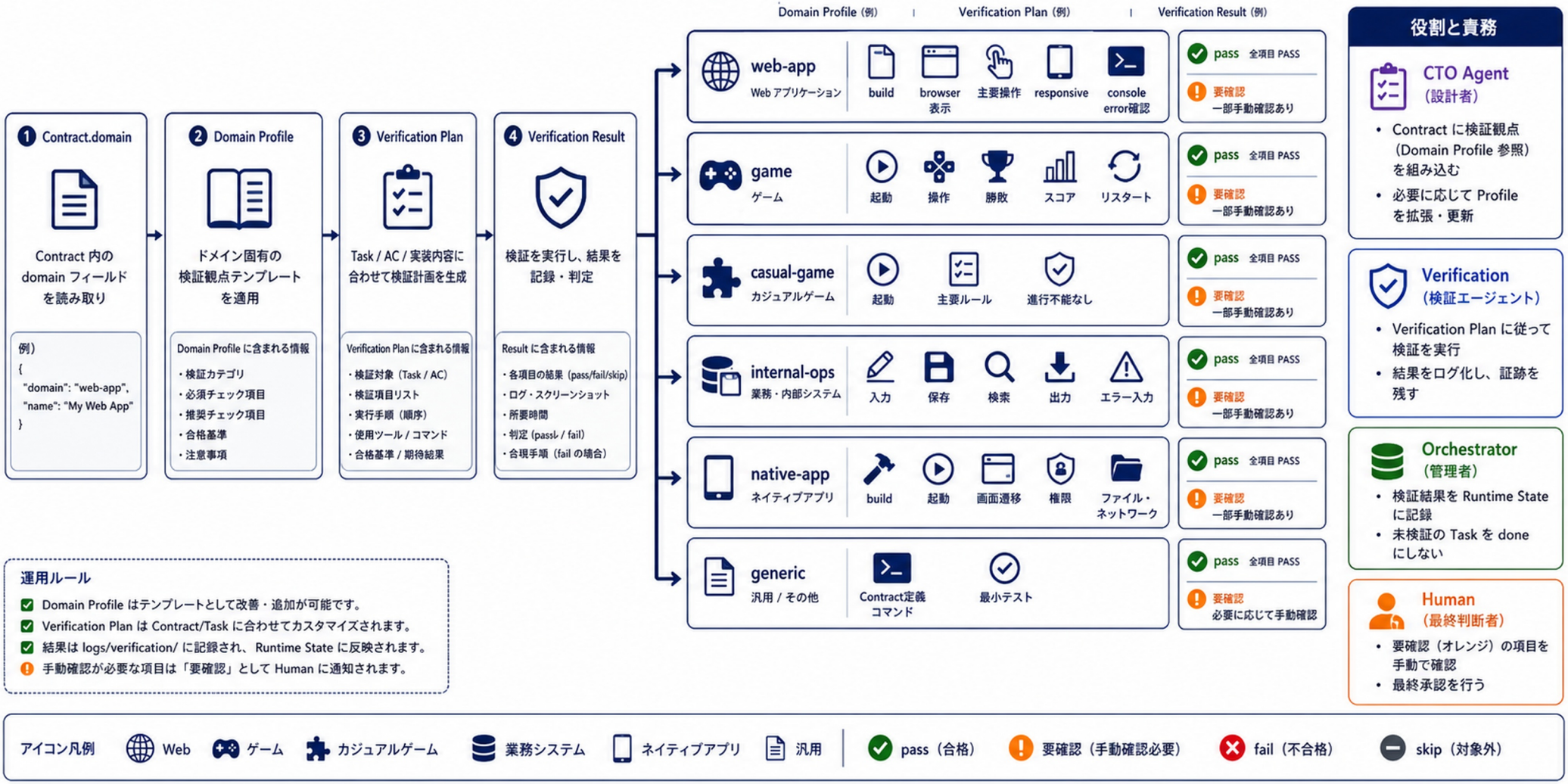
- 判断理由 / 現在状況
- 選択肢（2〜3個）と影響
- 推奨案
- Human への質問
- resume target（再開先）

凡例

- 検知の流れ
- 失敗ルート（分類）
- 正常リカバリ
- 停止 / 要対応
- 人間判断が必要
- 再開 / 復帰

16 16. ドメイン別検証選択（AI開発ハーネス）

Contract.domain に基づき、Domain Profile と検証計画を自動生成し、検証結果を記録します。



役割と責務

CTO Agent（設計者）
- Contract に検証観点（Domain Profile 参照）を組み込む
- 必要に応じて Profile を拡張・更新

Verification（検証エージェント）
- Verification Plan に従って検証を実行
- 結果をログ化し、証跡を残す

Orchestrator（管理者）
- 検証結果を Runtime State に記録
- 未検証の Task を done にしない

Human（最終判断者）
- 要確認（オレンジ）の項目を手動で確認
- 最終承認を行う

運用ルール

- Domain Profile はテンプレートとして改善・追加が可能です。
- Verification Plan は Contract/Task に合わせてカスタマイズされます。
- 結果は logs/verification/ に記録され、Runtime State に反映されます。
- 手動確認が必要な項目は「要確認」として Human に通知されます。

アイコン凡例　Web　ゲーム　カジュアルゲーム　業務システム　ネイティブアプリ　汎用　pass（合格）　要確認（手動確認必要）　fail（不合格）　skip（対象外）

17. MVP実装フェーズ （AI開発ハーネス MVP ロードマップ）

人間の「作りたい」から、MVPとして完了サマリーまで通せる最小機能を段階的に実装します。

Phase 1 — CLI基盤

- aih entrypoint
- path resolver
- atomic JSON save
- events.jsonl 記録
- aih init 実装

主担当： Codex

完了条件：
aih init でプロジェクトが初期化できる

Output artifact：
project.json / events.jsonl

Phase 2 — スキーマと状態

- project / coo / contract / tasks / runtime 各スキーマ
- 状態遷移の定義
- status / next / counters
- doctor コマンド実装

主担当： Codex

完了条件：
Runtime State が妥当性検証できる

Output artifact：
各 schema / doctor 結果

Phase 3 — 要件と契約

- prompt coo
- import coo-spec
- approve coo-spec
- prompt cto
- import contract
- approve contract

主担当： Codex

完了条件：
COO Spec / Contract を取り込み、承認できる

Output artifact：
coo_spec.md / contract.md / tasks.json

Phase 4 — 実装レビュー検証

- prompt implementer
- import implementation
- prompt reviewer
- import review
- verify（テスト・ビルド）
- bug report（記録）

主担当： Codex

完了条件：
レビューと検証のループを回して done にできる

Output artifact：
implementations/ / reviews/ / logs/verification/

Phase 5 — 人間判断

- decision create
- decision answer
- 上限停止（上限超過）
- 契約外変更停止

主担当： Codex

完了条件：
Decision の作成・回答・再開ができる

Output artifact：
decisions/ / decision.json / decision.answer.json

Phase 6 — 成果物とE2E

- summary 生成
- approve final
- artifacts manifest
- Todo Web App E2E フロー

主担当： Codex

完了条件：
「作りたい」から完了サマリーまで E2E で通る

Output artifact：
summary.md / manifest.json / レポート一式

マイルストーン

1. CLIで初期化できる — aih init 完了
2. 状態の整合性が取れる — doctor OK
3. 契約を取り込み承認できる — COO / Contract OK
4. レビューと検証のループを回して done にできる
5. 人間判断の介在で安全に進められる
6. E2Eで完了サマリーまで通せる

MVP完了 = Humanの「作りたい」から完了サマリーまで通せる

フェーズの考え方

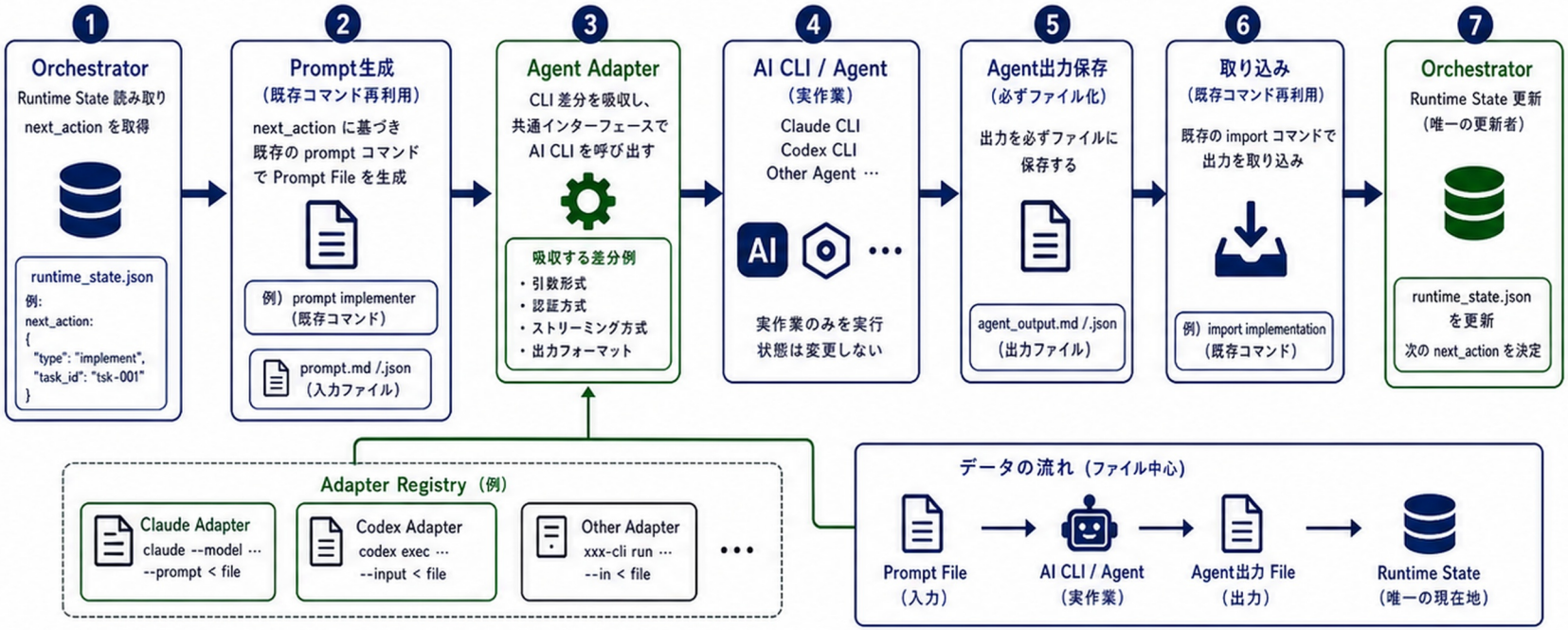
- 左から右へ段階的に完成度を上げる
- 各フェーズで「小さく完了」を積み重ねる
- 早期に E2E を回せることを最優先

凡例

- 基盤・設計
- 実装・検証
- 人間判断・ガード

18. 将来のAI自動起動（AI開発ハーネス）

Orchestrator が次アクションを判断し、AI CLI を自動で起動・取り込み・状態更新までを一気通貫で実行する将来像



役割と責務

Orchestrator
- Runtime State 読み取り・判定
- Prompt生成（既存コマンド呼出）
- Agent出力の取り込み
- Runtime State 更新（唯一）
- 次の next_action を決定

Agent Adapter
- AI CLI の差分を吸収
- 共通インターフェースを提供
- 認証・引数・出力形式の変換

AI CLI / Agent（Claude / Codex / その他）
- 実作業のみを実行
- 状態は変更しない
- 出力はファイルに書き出す

Runtime State（runtime_state.json）
- 唯一の現在地・唯一の更新者
- 全判断の根拠データ

Human
- 必要時に判断・承認
- Decision Packet への回答
- 最終承認（approval final）

運用コマンド（将来の自動起動用）

aih run-agent --role <role> --task <task_id> --project <project_id>

例） aih run-agent --role implementer --task tsk-001 --project prj-x

指定した role / task に対して、Prompt生成 → AI実行 → 取り込み → 状態更新までを

aih run-next --project <prj-x>

例） aih run-next --project prj-x

runtime_state.next_action を読み取り、次のステップを自動実行

設計原則（Key Principles）

既存promptファイルを入力にする
既存の prompt コマンドで生成した Prompt File をそのまま使用する

AI CLI差分はAdapterに閉じ込める
CLI ごとの違いは Adapter が吸収し、Orchestrator は共通インターフェースで制御

Agent出力は必ずファイル保存
出力を必ずファイル化し、取り込みを経てから状態を更新する

状態更新はOrchestratorだけ
runtime_state.json の更新は Orchestrator のみが行う（唯一の更新者）

run-agent / run-next は後続拡張
将来的にスケジュール実行、並列実行、通知連携などを拡張可能

この仕組みが整うことで、AI が自律的に次の作業を判断し、実行・検証・完了までを自動で進められます（Human は必要な判断のみを行う）。