# Agentic Project Plan — Framework

> This is a proposed plan and vision. The content reflects current thinking and is subject to change as the project evolves, requirements are clarified, and decisions are made with the team.

# 1. Vision & Goals

> **One repository, two kinds of code: the platform (built once, reused everywhere) and the projects (composed per use case).**

This framework defines how to build, structure, and operate an agentic platform that hosts multiple use cases within a single monorepo. The platform's purpose is to eliminate duplication across use cases by providing reusable, audit-grade building blocks that any project pod can compose without reimplementing.

**North-star metric:** The third use case onboards in under 2 sprints, reusing the platform with zero copy-paste.

## Goals

- Deliver agentic automation for domain-specific workflows that is explainable, auditable, and deterministic where rules apply
- Establish a reusable platform so each new use case costs less than the last
- Enforce governance (PII, data residency, role-based access) uniformly across all projects
- Give operators full observability into cost, latency, and agent decisions

---

# 2. Scope & Use Cases

## In Scope

| Use Case | Project Folder | Status |
|---|---|---|
| *(Use Case 1)* | `projects/<use-case-1>/` | *(Active / Planned)* |
| *(Use Case 2)* | `projects/<use-case-2>/` | *(Active / Planned)* |
| *(Use Case N)* | `projects/<use-case-n>/` | *(Planned)* |

> Populate this table with the actual use cases for the engagement.

## Out of Scope

- General-purpose chatbot or Q&A outside the defined domain
- Direct end-user UI (the platform exposes APIs; front-end is owned by consuming teams)
- Use cases outside the agreed domain boundary

---

# 3. Client Requirements

## What Clients Want

| Category | Requirement |
|---|---|
| **Comprehensiveness** | Deep, well-reasoned answers — not just a result, but *why* |
| **Explainability** | Full audit trail of agent decisions at every workflow node |
| **Flexibility** | Freedom to switch context mid-conversation without losing state |
| **Low Latency** | Streaming output where possible; animated traceability (live progress, not black box) |
| **HITL** | Human-in-the-loop escape hatches at every critical step |
| **Deterministic Rules** | Business rules win over LLM judgment (thresholds, approval logic, domain constraints) |

| Category | Requirement |
|---|---|
| **Guardrails** | PII blocked, hallucinations caught, out-of-scope actions rejected |
| **Role-Based Access** | Agent acts only within the calling user's permissions |
| **Graceful Degradation** | Uncertain? Fall back cleanly — never silently wrong |
| **Idempotency** | Same input → same output; no duplicate or ghost transactions in backend systems |
| **Cost Visibility** | Token usage per run/workflow/use case; chargeback-ready |
| **Versioned Artifacts** | Prompts and agent profiles rollback when a model update breaks behavior |
| **Eval Gate** | Golden cases must pass before any deployment |
| **Compliance Evidence** | Output exportable, signed, timestamped for audit |
| **Data Residency** | Model calls stay in-region; no data leaves approved boundaries |
| **Incremental Adoption** | Run alongside existing processes before full cut-over |
| **No New Portal** | Works inside existing tools (chat, ticketing, ERP) — not a separate portal to log into |

# What Clients Say vs. What Actually Blocks Projects

| They say | They block on |
|---|---|
| Low latency | Explainability (audit requirement) |
| Smart AI | Deterministic rules respected |
| Traceability | Cost |
| Automation | HITL escape hatch |

---

# 4. Architecture Overview

## Monorepo Zones

| Zone | Folders | Changes | Owned by |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **Platform** | `packages/`, `scripts/`, `infra/shared`, `docs/standards` | Slowly, carefully | Central Platform Team |
| **Project** | `projects/`, `apps/`, `evals/`, `infra/<project>`, `docs/runbooks/` | Frequently | Project Pods |

# Dependency Rules

```
apps/*  →  projects/*  →  packages/*
```

| Rule | Detail |
|---|---|
| ✅ `apps/*` may import `projects/*` and `packages/*` | |
| ✅ `projects/*` may import `packages/*` | |
| ✅ `packages/*` may import lower-level `packages/*` | |
| ❌ `packages/*` importing `projects/*` or `apps/*` | Circular — forbidden |
| ❌ One project importing another project directly | No pod-to-pod coupling |
| ❌ Agents calling integration clients directly | Must go through `enterprise_tools` |
| ❌ Workflows calling raw SAP/SQL directly | Must go through tool functions |

# Platform Response to Client Requirements

| Client Need | Platform Mechanism |
|---|---|
| Explainability | Workflow trace stored in persistent store; every node logs input/output |
| HITL | `HITLExecutor` node in every workflow; notification channel handoff (e.g. chat/email/ticket) |
| Deterministic rules | `*_decision_service.py` enforces rules before LLM can act |
| Cost visibility | Token tracking middleware on the LLM client factory |
| Versioned prompts | `configs/prompts/` and `configs/agent_profiles/` under source control |
| Eval gate | `evals/<project>/` golden cases run in CI before merge |
| Data residency | LLM client bound to approved in-region endpoint |
| Graceful degradation | `Default()` edges in every switch node; fallback executor required |
| Idempotency | Workflow inputs hashed; duplicate detection in worker entrypoint |

# 5. Code Structure

## Top-Level Layout

```
agent-platform/
  packages/                  # Platform zone — reusable, owned by Central Team
    agent_factory/           # Agent framework wrappers: load_declarative_agent,
create_code_agent
    enterprise_tools/        # @tool-decorated async functions for external system
integrations
    shared_models/           # Canonical Pydantic models and snapshot schema
    shared_utils/            # Logging, retry, config helpers
  projects/                  # Project zone — one folder per use case
    <project>/
      workflow/
        <project>_workflow.py      # WorkflowBuilder wiring (Graph API)
        messages.py                # Pydantic input/result/output types
        workflow_config.yml        # workflow_api: graph
        executors/
          <name>_executor.py       # Executor subclasses with @handler
      agents/
        <project>_agents.py        # Agent construction via agent_factory
      tools/
        <project>_tool.py          # @tool-decorated async functions (project-specific)
      skills/
        SKILL.md                   # MAF Skills stub
        references/
      configs/
        agent_profiles/            # kind: Prompt YAML files
        prompts/                   # System prompt .md files
        rules/                     # Deterministic rule definitions
      services/
        <name>_decision_service.py # Business rule enforcement
      tests/
        test_workflow.py
      evals/
```

```
  apps/                     # Worker entrypoints — one per project
    <project>-worker/
      main.py
      Dockerfile
  evals/                    # Cross-project eval runner
  infra/                    # IaC — shared + per-project
    shared/
    <project>/
  docs/
    standards/              # Platform coding standards (this file lives here)
    runbooks/               # Per-project operational runbooks
  scripts/                  # CI helpers, code generators
  ci_matrix.yml             # Change-aware CI: only test affected projects
  CODEOWNERS                # Enforces ownership boundaries
```

# Agent & Workflow Design Standards

- **Graph API is production default** — `WorkflowBuilder`, `Executor` subclass + `@handler`, `IWorkflowContext.yield_output()`
- **Functional API (`@workflow/@step`) is prototyping only** — never in production
- Every workflow must have a `Default()` fallback edge on every switch node
- Every workflow must have a `HITLExecutor` node for decisions above a confidence threshold
- Agents are created via `agent_factory` — never instantiated directly from MAF classes
- Tools are plain `async` functions decorated with `@tool` — no `BaseTool` class

## Declarative Agent YAML Schema

```
kind: Prompt
name: "agent_name"
description: "..."
instructions: "..."
model:
  id: "gpt-4o"
  provider: "AzureOpenAI"
  apiType: "azure"
  options: { temperature: 0, max_tokens: 2048 }
# outputSchema: optional JSON Schema for structured output
```

---

# 6. Integration Points

| System | Package | Access Pattern |
|---|---|---|
| **ERP / Backend System** | `packages/enterprise_tools/<system>/` | `@tool` async functions over system API |
| **Ticketing / ITSM** | `packages/enterprise_tools/<itsm>/` | REST API client wrapped in `@tool` |
| **Document / Search** | `packages/enterprise_tools/search/` | `@tool` async search functions |
| **LLM Provider** | `packages/agent_factory/` | LLM client via `AgentLLMFactory` |
| **Session Store** | framework `HistoryProvider` | Session state and workflow checkpoints |
| **Notification Channel** | `apps/<project>-worker/` | HITL handoff (chat card, email, ticket) |

> Replace placeholder names with actual systems for the engagement.

**Rule:** Project pods never instantiate integration clients directly. All external calls go through `packages/enterprise_tools/`.

---

# 7. Testing & Eval Strategy

## Layers

| Layer | Location | What it tests | Runs when |
|---|---|---|---|
| **Unit** | `projects/<p>/tests/` | Individual executors, tools, decision services | Every PR |
| **Integration** | `projects/<p>/tests/` | Workflow end-to-end with mocked external calls | Every PR |
| **Golden-case Eval** | `evals/<p>/` | Real LLM calls against known inputs/outputs | Pre-merge gate |
| **Regression** | `evals/<p>/` | Score must not drop below threshold vs. baseline | Pre-merge gate |
| **Cross-project** | `evals/` | Shared `packages/` change runs all affected project evals | On `packages/` PR |

## Eval Standards

- Every project must have a minimum of 10 golden cases before going to production
- Eval score threshold: ≥ 90% pass rate required to merge
- Golden cases cover: happy path, edge cases, HITL trigger conditions, rule enforcement
- Evals run against the same LLM endpoint as production (no mocked LLM in evals)

## Testing Methods

- **Executor unit tests** — instantiate executor, call handler directly, assert on `yield_output` captures
- **Workflow integration tests** — build full graph, inject mock tool responses, assert final output model
- **Decision service tests** — pure Python, no LLM, 100% deterministic, must have 100% coverage
- **Eval runner** — `scripts/run_evals.py --project <name>` scores against golden cases

---

# 8. Observability & Cost

## Tracing

- Every workflow node logs: `node_id`, `input_hash`, `output_summary`, `latency_ms`, `token_usage`
- Traces stored in persistent store; queryable per `session_id` and `workflow_run_id`
- Structured logs emitted to the agreed observability platform (e.g. Azure Monitor, Datadog, ELK)

## Cost Tracking

- LLM client middleware captures prompt tokens, completion tokens, model ID per call
- Cost rolled up per: individual call → executor → workflow run → project → daily aggregate
- Chargeback tags: `project`, `use_case`, `environment`

## Dashboards *(stub — to be built in Sprint 2)*

- Cost per workflow run (by project)
- P50/P95 latency per executor node

- HITL trigger rate (% of runs requiring human intervention)
- Eval pass rate trend over time

## Alerting *(stub)*

- Cost spike: > 2× 7-day average triggers on-call alert
- Eval regression: < threshold on any project blocks deployment pipeline

---

# 9. Governance & Compliance

## CODEOWNERS Enforcement

- `packages/` → Central Platform Team; any PR requires Platform Architect approval
- `projects/<p>/` → Pod Tech Lead; cross-pod changes require both pod leads
- `CODEOWNERS` is the physical enforcement of team boundaries — it is not optional

## Data & Security

- All model calls use an LLM client bound to an approved in-region endpoint
- No data leaves the approved tenant/cloud boundary
- PII detection runs as a guardrail before any data is sent to the LLM
- Role-based access: agent inherits the permissions of the calling user's identity

## Audit Evidence

- Every workflow output is signed with `workflow_run_id`, `timestamp`, `user_id`, `model_version`
- Outputs exportable as JSON for compliance review
- Audit log retained for the agreed retention period (minimum 90 days)

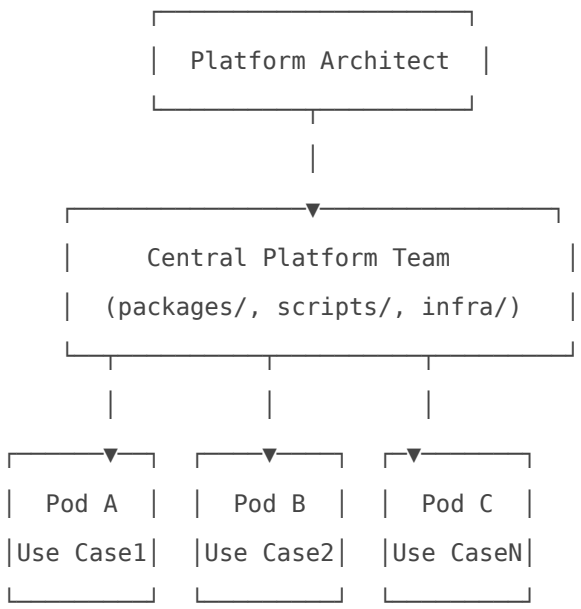
## Change Governance

- Any new shared abstraction in `packages/` requires Platform Architect sign-off
- Project-specific logic must not creep into `packages/` — the Architect says no
- Model/prompt version changes require a passing eval run before deployment

---

# 10. Team Formation & Ownership

## Hub-and-Spoke Structure




## Central Platform Team (owns `packages/`)

| Role | Responsibility |
|---|---|
| Platform / Agent Engineer | Agent framework wrappers, agent factory, tool registry standards |
| Integration Engineer | External system tools (ERP, ITSM, search, document processing) |
| Domain / Data Modeler | Canonical Pydantic models, snapshot schema |
| Eval & Governance Engineer | Eval framework, golden-case runner, PII/guardrails |
| DevOps Engineer | Base image, change-aware CI, KEDA templates |

Typical size: **3–5 people.** Protect this team's capacity — if it's a bottleneck, every pod slows down.

## Project Pods (own `projects/<p>/` + 4 sibling folders)

| Role | Responsibility |
|---|---|

| | |
|---|---|
| Workflow Owner / Tech Lead | Composes the workflow graph; node order, branching, HITL |
| Prompt / Agent-Profile Engineer | Project prompts + agent profiles (reuses central agents) |
| Rules / Business Analyst | Deterministic rules + decision policy |
| Tools Engineer | Composes central tools into project-specific tools |
| QA / Eval Owner | Golden cases, regression thresholds |
| App / Deploy Owner | Worker entrypoint, queue, scaling, Dockerfile |

Typical size: **2–4 people per pod** (roles double up on small pods).

# Staffing Summary

| Scenario | Composition | Headcount |
|---|---|---|
| Lean (roles doubled) | 1 architect + 2 central + 2/pod × 3 pods | ~9 |
| Comfortable | 1 architect + 4 central + 3/pod × 3 pods | ~14 |

# Rules of Engagement

- Pods import `packages/*` — they do not edit it. Need a change? Raise a PR to the Central Team
- No pod imports another pod
- The Platform Architect approves any new shared abstraction
- Project-specific logic must never creep into `packages/`

---

# 11. Delivery Milestones

| # | Milestone | Success Criteria |
|---|---|---|
| 1 | **Platform Baseline** | `packages/` builds, CI green, one golden eval passes end-to-end |
| 2 | **First Project Live** | End-to-end workflow deployed; HITL tested in notification channel; cost dashboard live |
| 3 | **Second Project Onboards** | Reuses platform cleanly — no copy-paste, no `packages/` edits required |

| # | Milestone | Success Criteria |
|---|---|---|
| 4 | **Reuse Proven** | Third project onboards in < 2 sprints using existing platform |
| 5 | **Full Observability** | Cost, latency, and eval dashboards live; alerting configured |

> **The second project is the real test.** Clean reuse proves the platform. Friction proves the abstraction is wrong.

# Weekly Cadence

| Ritual | Who | Purpose |
|---|---|---|
| Platform sync | Architect + pod leads | Pods request shared capabilities; central prioritizes |
| Shared-change review | Central + affected pods | Any `packages/` change runs all affected project evals |
| Reuse checkpoint | Architect | When pod #2 onboards: can it reuse the platform cleanly? |

---

# 12. Development Lifecycle

> Eval iterations run *throughout* Development — step 4 is the exit gate, not the first time evals run.

| # | Phase | Key Activities | Exit Criteria |
|---|---|---|---|
| 1 | **Discovery & Requirements** | Business process mapping; identify decision points; define rule vs. LLM boundaries; gather initial golden examples | Process map signed off; rule/LLM boundary agreed; ≥ 10 golden input/output pairs captured |
| 2 | **Design** | Workflow graph (nodes, branches, HITL triggers); agent profile design; data model; integration contracts | Workflow design reviewed; data model approved; integration specs agreed |

| # | Phase | Key Activities | Exit Criteria |
|---|---|---|---|
| 3 | **Development** *(with ongoing eval iterations)* | Build executors, tools, agents, decision services, prompts; run eval loop continuously; iterate prompts against golden cases | All executors implemented; decision services have 100% test coverage; eval loop stable |
| 4 | **Unit & Integration Testing + Eval Gate** | Executor unit tests; end-to-end workflow with mocked integrations; eval threshold check | ≥ 90% eval pass rate; unit/integration tests green in CI |
| 5 | **SIT** *(staging deploy)* | Deploy to staging; end-to-end on real/staging backends; HITL path exercised; idempotency verified | All workflow paths pass on staging; HITL handoff confirmed; no duplicate transactions |
| 6 | **QA, Performance & Cost Baseline** | Latency profiling per node; token cost per run; UX review; set alert thresholds; PII/guardrail validation | Latency within SLA; cost per run baselined; UX sign-off; security review passed |
| 7 | **UAT** | Business stakeholders validate golden cases + real production scenarios; HITL UX accepted | Business sign-off; UAT defects resolved or deferred with owner |
| 8 | **Go Live** | Production deploy; eval gate re-run on prod config; smoke test; runbook ready; on-call briefed | Prod smoke test passes; dashboards live; on-call trained |
| 9 | **Hypercare** *(minimum 4–6 weeks)* | Monitor cost/latency dashboards; rapid prompt rollback if needed; capture new golden cases from real failures; tune alert thresholds | Eval pass rate stable; no P1 incidents; golden case set updated with production learnings |

# Key Differences from Standard SDLC

| Standard SDLC | Agentic Adaptation |
|---|---|
| Testing is a phase after development | Eval runs continuously inside development (step 3) |
| Bugs are deterministic — fix the code | Regressions can be non-deterministic — fix the prompt, re-eval, re-gate |
| UAT validates features | UAT validates *judgment* — does the agent decide correctly on real cases? |
| Go Live is the finish line | Go Live opens the hypercare window — production edge cases are expected |

| Standard SDLC | Agentic Adaptation |
|---|---|
| Hotfix = code change | Hotfix can be a prompt update — must still pass eval before deploy |

# 13. Risks & Mitigations

| Risk | Likelihood | Impact | Mitigation |
|---|---|---|---|
| Platform becomes a bottleneck | Medium | High | Protect Central Team capacity; pods can draft PRs, central reviews |
| Model version drift breaks evals | High | Medium | Pin model versions in `configs/agent_profiles/`; eval gate on every deploy |
| Project-specific logic creeps into `packages/` | Medium | High | CODEOWNERS + Architect veto; enforce at PR review |
| External API changes break tools | Medium | High | Integration tests mock at HTTP layer; version-pinned API specs |
| HITL adoption — users bypass the approval step | Low | High | Notification times out → escalation; no auto-approve after timeout |
| LLM hallucination on rule-bound decisions | Medium | High | Decision service enforces rules deterministically before LLM output is acted on |
| Data residency violation | Low | Critical | LLM client region-locked; infra reviewed by Security before go-live |
| Eval golden cases become stale | Medium | Medium | QA/Eval Owner reviews cases each sprint; flag if business rules change |

*Related: Architecture Decision Records · Agent Standards · Eval Strategy · CI/CD Pipeline*

Revision #1
Created 2026-07-01 15:07:01 UTC by Soubhik Mazumdar
Updated 2026-07-01 15:13:28 UTC by Soubhik Mazumdar