OhanaSecure

# AI Fluency Curve

## 5 Phases of Organizational AI Maturity

From first prompt to cost-governed multi-LLM production systems

SDD Overlay included · Spec-Driven Development Levels 1 – 3

For internal teams — understanding where you are in the curve

# The Framework

Two overlapping dimensions tell you exactly where you stand

## AI FLUENCY PHASES

**01 Chat & Prompt Engineering**
Individual tool usage, prompt iteration, human output review

**02 Agentic**
Single agent with tools, memory, and goal-directed behavior

**03 Harnessed / Councils**
Multi-agent orchestration, security harnesses, agent registries

**04 Agentic Loops**
Self-correcting loops, step functions, eval-retry-escalate patterns

**05 Cost-Governed Multi-LLM**
Model routing, fallbacks, token budgets, ephemeral execution

## SDD — SPEC-DRIVEN DEVELOPMENT

**L1 No Spec**

Vibe-coding. Instructions live in chat history only. Nothing survives a session reset.

**L2 Spec in Repo**

Agent instructions and goals are committed markdown files. Reproducible and versionable.

**L3 Spec as Source**

The spec IS the product. Agents build from it autonomously. CI/CD enforces conformance.

*Both dimensions are independent — a team can be Phase 3 with L1 SDD. That gap is a risk vector.*

# 01 Phase 1 · Chat & Prompt Engineering

SDD L1 · No Spec

Using AI as a smart search engine or co-writer. Output is reviewed and copy-pasted by humans.

## KEY CONCEPTS & CAPABILITIES

### Prompt Construction

System + user + few-shot examples. Role prompting, chain-of-thought, output formatting.

### Context Window Awareness

Token limits, context degradation, when to start a fresh session.

### Output Evaluation

Human reviews every output. No automation — trust but verify.

### Manual Tool Chaining

Copy-paste between tools. Human feeds one AI's output into another.

### Model Selection Basics

GPT-4 vs Haiku vs Sonnet. Cost vs capability tradeoff at the human level.

## OS REPO & HARNESS EXAMPLES

### agent/navi-riya.md

Prompt-only Navi instructions. Markdown defines persona and guardrails.

### days/day-08 — Structured Prompting

Riya's Day 8 curriculum: templates, role framing, output constraints.

### team/my-outputs/prompt-library.md

Manually curated prompt templates — the canonical Phase 1 artifact.

### agent/day-05-jc-finding.md

Redacted JC finding shared as prompt context. Human curates what AI sees.

### agents/instructions/*.md (flat)

Session-prep and followup prompts — no tool wiring yet.

## COMMONLY MISSING

No version control on prompts · No retry logic · No output schema · No eval loop · Prompts live in chat, not repo

02

# Phase 2 · Agentic

SDD L2 · Spec in Repo

A single agent with tools, memory, and persistent goals. Humans review final output, not every step.

## KEY CONCEPTS & CAPABILITIES

### Tool Use (Automated)

Agent calls APIs, reads files, writes to databases without human hand-holding.

### Memory & State

Short-term (context), long-term (SSM / Airtable). Agent remembers across sessions.

### Goal-Directed Loops

Perceive → decide → act → observe. Loop runs until goal is achieved or halted.

### Agent Identity

Named identity (Navi, Vigil, Stark). Instructions define role, scope, refusal patterns.

### Async Delegation

Humans delegate tasks and check results later. Not every step needs approval.

## OS REPO & HARNESS EXAMPLES

### slack-agent/inbound.js

Slack → Bedrock dispatch loop. Mentions trigger agent execution via Step Functions.

### agents/agent-registry.json

Registry of all named agents: Navi, Vigil, Stark, Sherlock, n8n. Identity + scope.

### /ohanasecure/agents/*/identity-token

Per-agent SSM identity tokens provisioned by setup-agent-registry workflow.

### agents/instructions/session-prep.md

Session Prep: reads Airtable, synthesizes intake, produces briefing autonomously.

### agents/instructions/booking-followup.md

Post-session 4-step email sequence triggered by closeout. Autonomous multi-step.

## COMMONLY MISSING

No cost cap per agent · No scope enforcement between agents · No audit log · Shared credentials · No eval harness

03

# Phase 3 · Harnessed & Councils

SDD L2 · Spec in Repo

Multiple agents coordinated under governance. Security harnesses enforce boundaries. Councils debate before acting.

## KEY CONCEPTS & CAPABILITIES

### Security Harness (JC)

Jean-Claude pipeline: agents run inside constraints. Every action scanned before execution.

### Agent Councils

Multiple LLMs review the same task. Majority vote or judge model decides.

### Scope Enforcement

Agents access only what they're scoped to. Fail-closed: unknown scope = no token.

### Token Broker Pattern

Agents request scoped tokens from a broker. No standing credentials. Ephemeral access.

### Injection Protection

Input sanitization on every inbound message. Prompt injection neutralized at ingest.

## OS REPO & HARNESS EXAMPLES

### token-broker/handler.js

Enforces permitted_scopes per agent. Fail-closed — no permitted_scopes = 403.

### agents/harness — RALPH council

Pentest harness: Opus + Fable + GPT council attacking same target simultaneously.

### setup-agent-registry.yml

CI/CD syncs agent-registry.json → SSM. Infrastructure-as-Code for agent identity.

### iam/n8n-broker-invoker.yaml

Least-privilege IAM: n8n can only invoke the broker Lambda. Nothing else.

### inbound.js — sanitizeInput()

Allowlisted channel enrollment + injection protection at the front door.

## COMMONLY MISSING

No harness eval scoring · Council decisions not in audit trail · No model version pinning · Scope changes require manual update

04

# Phase 4 · Agentic Loops

SDD L3 · Spec as Source

Self-correcting execution with structured eval, retry, escalation, and halt. Minimal human intervention.

## KEY CONCEPTS & CAPABILITIES

### Step Function Orchestration

Multi-step agent workflows with branching, retries, parallel execution. AWS SFN.

### Eval → Retry → Escalate

Agent evaluates its own output, retries with different approach, escalates if stuck.

### Halt Conditions

Explicit stop: budget exceeded, confidence too low, N retries exhausted, human gate.

### Structured Output Contracts

JSON Schema enforced on every agent output. Downstream agents consume typed data.

### Feedback Loops

Navi reviews its own prior outputs (feedback cron). Agent learns from flagged errors.

## OS REPO & HARNESS EXAMPLES

### ohanasecure-agent-loop-prod (SFN)

Slack mention → SFN state machine → Bedrock → tool execution → Slack reply.

### navi-riya-feedback-cron

Daily: fetches Riya's commits → reviews content → posts coaching to Slack.

### booking-followup/handler.js

Idempotent daily loop: scans records, checks time windows, marks flags. Retry-safe.

### probes/session-lifecycle-live.js

E2E probe with mode-guard: synthetic booking traverses full loop. Blocks real sends.

### RALPH halt pattern

Build-failure triggers halt, fixer loop attempts repair, escalates to #ops if stuck.

## COMMONLY MISSING

No distributed tracing · Retry budget not globally enforced · No eval scoring trend storage · Halt escalation Slack-only — no page

05

# Phase 5 · Cost-Governed · Multi-LLM · Ephemeral

SDD L3 · Spec as Source

Production-grade AI. Token budgets enforced. Models swapped dynamically. Execution is ephemeral by design.

## KEY CONCEPTS & CAPABILITIES

### Model Routing & Fallback

Haiku for fast ops, Sonnet for reasoning, Opus for council. Auto-fallback on quota.

### Token Budget Governance

Per-agent, per-task caps at the broker layer. Cost overruns trigger halt, not degraded output.

### Ephemeral Execution

Stateless runs. State lives in SSM, Airtable, or S3 — never in-process.

### Multi-LLM Arbitration

Different models produce outputs; an arbiter selects the best. Prevents single-model failure.

### Observability & Attribution

Every inference logged: model, tokens, cost, agent ID, task ID. Per-agent dashboards.

## OS REPO & HARNESS EXAMPLES

### claude-haiku-4-5 (Bedrock default)

Default Navi model. Haiku selected for speed + cost at Phase 5 governance.

### RALPH — Opus + Fable + GPT council

Pentest harness routes to 3+ models per task. Each model run scoped and billed.

### token-broker — fail-closed scoping

Short-lived tokens per invocation. No standing access — true ephemeral credential.

### /agents/*/identity-token per-agent

Isolated identity per agent. No shared credentials across the agent fleet.

### Perplexity + n8n hybrid brief

Daily brief: Perplexity fires it, n8n delivers it. Cost split across two runtimes.

## COMMONLY MISSING

No centralized cost dashboard · Model drift detection manual · No cost-per-outcome metric · Lambda /tmp PII audit missing

# SDD Overlay — Spec-Driven Development

How mature is your relationship with specifications? This dimension cuts across all 5 phases.

## L1 · No Spec

### WHAT IT IS

Instructions live in the chat window. Every session starts from scratch. If the prompter leaves, the knowledge leaves.

### SIGNALS

Ad-hoc prompts, no version history, output varies wildly, no shared vocabulary for AI tasks.

### RISK

CRITICAL — No reproducibility. Cannot audit, improve, or hand off to another person.

### OS EXAMPLE

Pre-registry era. Prompts lived in individual sessions — no commit, no review, no trace.

## L2 · Spec in Repo

### WHAT IT IS

Agent instructions and goals are committed markdown files. Human writes the spec, agent reads it.

### SIGNALS

agents/instructions/*.md, agent-registry.json, navi-riya.md, cron task definitions.

### RISK

MEDIUM — Spec drift. Code changes but spec lags. No enforcement that agents follow the spec.

### OS EXAMPLE

Current state: agent-registry, booking-followup.md, session-prep.md, all riya-sandbox day files.

## L3 · Spec as Source

### WHAT IT IS

The spec IS the deliverable. Agents build artifacts from spec autonomously. CI/CD validates conformance.

### SIGNALS

Workflows triggered by spec changes, eval harnesses testing output vs spec, schema-validated artifacts.

### RISK

LOW (if harness is good) — Risk shifts to spec quality. Bad spec = bad output at scale.

### OS EXAMPLE

setup-agent-registry.yml syncs spec→SSM. deploy-* workflows enforce infra from spec. Wave 16 probes validate.

# The Foundation Stack

*OSI-inspired*

Agentic systems don't create enterprise-grade quality. The infrastructure beneath them does. Every weakness at L1–L5 is inherited by every agent above it — at machine speed.

| | | | |
|---|---|---|---|
| L7 | **Agentic Systems** | Agents · Harnesses · Councils · Multi-LLM · Eval loops · Ephemeral execution | *The phases. Only stable when L1–L6 are solid.* |
| L6 | **LLM Infrastructure** | Model routers · Token budgets · Fallback chains · Bedrock/API abstraction · Per-agent cost attribution | *New layer — most orgs don't have it yet.* |
| L5 | **Data & State** | Stateless compute · Managed state stores (SSM, S3, RDS) · PII controls · Idempotency · Audit trail | *Agents must never hold state in-process.* |
| L4 | **Observability** | Structured logging · Distributed tracing · Cost metrics · Per-agent dashboards · Alerting · Anomaly detection | *Without this, you don't know what your agents are doing.* |
| L3 | **Compute & Runtime** | Serverless (Lambda) · Containers (ECS/Fargate) · Ephemeral-by-default · No long-lived processes · Auto-scaling | *Agents need ephemeral, stateless, scoped compute.* |
| L2 | Identity & Secrets | Least-privilege IAM · Secrets stores (SSM/Vault) · No hardcoded creds · Token broker · Per-agent identity | *The most common failure point. Agents will use whatever access they can get.* |
| L1 | Network & Hardening | VPC · TLS everywhere · WAF · No port 80 · Patch management · OS hardening · Egress controls · DDoS protection | *The physical layer. Agents inherit every gap here.* |

⚠️ L6 and L7 are where AI lives. But L1–L5 are where AI fails. Most teams treat them as someone else's problem until an agent exfiltrates a secret, runs up a $40k inference bill, or produces unauditable output that can't be traced.

# Your Existing Infra May Need to Change

Services built for humans break differently under agentic load. Agents don't read docs, don't slow down, don't notice they're doing something wrong, and don't stop unless you make them.

| INFRASTRUCTURE AREA | BUILT FOR HUMANS — BREAKS WITH AGENTS | WHAT MUST CHANGE | RISK IF YOU DON'T |
|---|---|---|---|
| **Secrets & Credentials** | *Hardcoded API keys, .env files, shared service accounts. Humans rotate manually.* | SSM Parameter Store or Vault. Token broker per agent. No standing credentials. | **Agent leaks keys, exfiltrates secrets, or exceeds scope silently.** |
| **IAM & Access Control** | *Broad roles assigned to people. "DevAdmin" does everything.* | Per-agent least-privilege IAM. Fail-closed scoping. Agent registry enforces scope. | **One compromised agent has blast radius of an admin.** |
| **APIs & Service Design** | *Monolith endpoints, human-readable errors, rate limits tuned for browser traffic.* | Scoped Lambda endpoints. Structured JSON responses. Agent-safe rate limits + circuit breakers. | **Agent hammers endpoints, misreads errors, retries into a $10k Lambda bill.** |
| **Logging & Tracing** | *Application logs. CloudWatch. Someone checks dashboards when something breaks.* | Distributed tracing with agent ID, task ID, model, tokens, cost on every log line. | **Agent failure is invisible. You can't audit, replay, or improve what you can't trace.** |
| **Compute & Patching** | *Long-lived EC2s, manual patching cycles, containers reused across deploys.* | Ephemeral serverless/containers. Automated patching pipelines. Immutable infra. | **Agent runs in an unpatched environment. Vulnerabilities are now part of the attack surface.** |

*AI doesn't make bad infrastructure good. It makes bad infrastructure fail faster, at scale, with less visibility, and harder to roll back. The upgrade is not optional — it's the price of admission.*

# Where Are You?

Use these questions to place your team on the curve. Phase 2 with L2 SDD beats pretending to be Phase 4.

**P01**

### Do team members write structured prompts with system roles and output formats?

→ Move up when: an agent takes an action you didn't manually trigger.

**P02**

### Do you have a named agent with committed instructions in a repository?

→ Move up when: multiple agents need to coordinate without human handoffs.

**P03**

### Do your agents have enforced scope limits and a security harness?

→ Move up when: agents need to evaluate and retry their own outputs.

**P04**

### Do your agents self-correct and escalate on failure without paging you?

→ Move up when: you're making model and cost decisions per task, per agent.

**P05**

### Do you have per-agent cost attribution and model routing logic in code?

→ You're here. Focus: spec quality, eval harness rigor, and audit depth.

**SDD**

### If your best prompter left tomorrow, could your agents still run correctly?

→ L1→L2: commit prompts as markdown. L2→L3: let the spec generate the artifact.

My team today: Phase ___ SDD Level ___ Target in 90 days: Phase ___ SDD Level ___

# The Curve Is Not Linear

Most teams plateau at Phase 2 with L1 SDD. The leverage is in closing that gap — not chasing Phase 5.

### Start with spec

Commit your prompts and agent instructions before anything else. L2 SDD unlocks every phase above it.

### Scope before scale

Agents without enforced scope boundaries are liabilities. Harness first, expand access second.

### Govern cost per outcome

Token counts are vanity. Cost per delivered result is the metric that governs Phase 5 operations.