HITACHI

OFE Manifesto

Outcome-Forward Engineering

June 2025

Hitachi Digital Services

Table of contents

POWERING ENGINEERING DISCIPLINE

The OFE Manifesto

Writing more code won't get you there. Neither will adding agents or chasing the latest orchestration trend. **Outcome-Forward Engineering** means finding the shortest governed path from what you intend to what you can measure.

Most delivery models treat the outcome as the finish line. OFE treats it as the starting point. Not the ticket, not the tool, not the backlog. The outcome itself.

When it comes to agentic engineering, capability doesn't come from piling on tools, giant instruction files, or the framework everyone's talking about this month. It comes from giving agents exactly the context they need, writing clear task contracts, keeping research separate from implementation, verifying results, and owning the outcome. The blunt truth is that agents work best when humans do the hard thinking up front.

For OFE, that becomes the operating principle.

PRINCIPLE 1

We Engineer Outcomes, Not Activity

We don't count prompts written or agents launched. We don't care how many branches you created or ceremonies you sat through. We care whether the business outcome moved.

We measure progress by whether the business, operational, customer, or engineering outcome actually shifted.

Before starting anything, an OFE Engineer asks three questions: what outcome needs to change, how will we know it changed, and what's the smallest safe path to get there?

Code, agents, automation, architecture - all of these are tools. The outcome is what matters.

PRINCIPLE 2

We Start with Intent, Then Capability, Then Execution

Traditional delivery jumps from requirements to implementation. OFE moves from intent to outcome, but only after understanding what capability the system or team actually needs.

The sequence is:

OFE step	Engineering question
Intent	What is the organisation trying to achieve?
Outcome	What measurable result proves progress?
Capability	What ability must the system or team gain?
Contract	What must be true for the work to be complete?
Execution	What should humans, agents, services and platforms do?

Verification	What evidence proves the outcome has been achieved?
Learning	What should be retained as a reusable rule, skill or pattern?

This avoids the common agentic failure mode: asking powerful tools to execute poorly framed work.

PRINCIPLE 3

We Treat Context as an Engineering Asset

Context is everything, but more isn't better. Agents do better work when they get exactly what they need and nothing else.

So OFE Engineers curate context deliberately. We don't dump entire repositories, old documents, and historic decisions into every task. We build small, focused context packages with only what's relevant.

A good OFE context package contains:

Context component	Purpose
Outcome	The measurable result expected
Boundary	What is in scope and out of scope
Constraints	Security, compliance, architecture and delivery rules
Current state	The minimum system knowledge required
Decision record	What has already been decided and why
Verification contract	Tests, evidence, screenshots, metrics, review gates
Exit condition	What "done" means

An OFE Engineer isn't a prompt operator. They're a context architect.

PRINCIPLE 4

We Separate Research from Implementation

Tell an agent to 'build authentication' and it'll burn context figuring out what authentication even means, what options exist, which patterns apply, and how to implement it. Splitting research from implementation is important, so that execution starts with a decision already made and a clean slate.

OFE takes this as a delivery rule.

- Research produces options and trade-offs.
- Architecture turns those into decisions.
- Implementation executes the decision.
- Verification proves it worked.

These phases can be quick, but don't mix them. Merge research and implementation into one agent session and you get hallucination, drift, and accidental design.

PRINCIPLE 5

We Write Contracts Before We Delegate

Agents usually know how to start a task. They're worse at knowing when to stop. We need task contracts, tests, screenshots, and verification criteria as explicit end conditions.

OFE Engineers do the same.

> This task is complete only when:
> 1. The agreed behaviour is implemented.
> 2. The defined tests pass.
> 3. The agreed non-functional constraints are met.
> 4. The change is reviewed against architecture and security rules.
> 5. Evidence is attached.
> 6. The outcome impact is traceable.

A task without an exit contract isn't ready for agents.

PRINCIPLE 6

We Use Agents as Execution Partners, Not Accountability Substitutes

OFE Engineers can hand off research, implementation, refactoring, testing, documentation, migration analysis, code review, and operational analysis to agents. But they can't hand off accountability.

Agents can handle design and implementation, but humans still own the outcome. In OFE, this is non-negotiable.

The engineer remains accountable for:

Area	OFE accountability
Outcome	Is this solving the right problem?
Architecture	Does it fit the wider system?
Risk	What could fail, leak, drift or degrade?
Verification	What evidence proves it works?
Economics	Is the reasoning, compute and operational cost justified?
Operability	Can this run, recover and be supported?
Governance	Can decisions be explained and audited?

Agents make work faster. OFE Engineers make it safe, useful, and worth the cost.

PRINCIPLE 7

We Prefer Simple Systems Until Complexity Is Earned

We are sceptical of excessive harnesses, plug-ins, memory systems, and large instruction files. They bloat context and lock teams into solutions that models might outgrow.

OFE Engineers should be just as disciplined.

- We don't build orchestration theatre.
- We don't add an agent where a rule, script, workflow, or API call would work.
- We don't confuse architectural sophistication with business value.

The OFE hierarchy is:

> Manual clarity before automation.
> Automation before agents.
> Single agent before multi-agent.
> Contract before orchestration.
> Verification before autonomy.
> Outcome before platform.

Complexity has to earn its keep.

PRINCIPLE 8

We Build Reusable Skills, Not Reusable Noise

Traditionally rules encode preferences or constraints, while skills encode repeatable recipes for doing work.

OFE formalises this.

- Rules define how work must be done.
- Skills define how a capability is executed.
- Patterns define how teams repeatedly deliver outcomes.

A strong OFE engineering organisation will accumulate a governed library of:

Asset	Example
Rules	Security review rules, testing rules, coding standards
Skills	API design skill, migration assessment skill, PR review skill
Contracts	Prototype contract, production-readiness contract, incident-analysis contract
Patterns	RAG ingestion pattern, agent approval pattern, workflow automation pattern
Scorecards	Value, risk, readiness, cost and operability scoring
Evidence templates	Test evidence, architecture decision record, risk sign-off

The point isn't to make agents smarter by feeding them more text. It's to make delivery repeatable through better engineering assets.

PRINCIPLE 9

We Design for Verification from the Beginning

OFE Engineers don't ask 'does it look done?' They ask 'what evidence would convince a sceptical operator, architect, security lead, product owner, and finance owner?'

Verification belongs in the design, not tacked on at the end.

For OFE, evidence will include:

Evidence type	Example
Functional	Tests pass, expected behaviour demonstrated
Operational	Logs, traces, alerts, failure handling
Security	Threat model, access control, policy checks
Financial	Token, compute, storage and tool-call cost
Architectural	Decision record, dependency map, integration trace
User	Screenshot, workflow proof, acceptance test
Outcome	KPI movement, cycle-time reduction, error reduction

OFE Engineers don't ship assertions. They ship evidence.

PRINCIPLE 10

We Challenge Sycophancy and Design for Dissent

Agents tend to comply. They may give you the answer you seem to want, even when your premise is weak. It recommends neutral prompts and adversarial review to reduce bias.

OFE Engineers should make this standard practice.

Every important decision needs a challenge path. That means:

> Ask for findings, not agreement.
> Ask for trade-offs, not confirmation.
> Ask for risks, not reassurance.
> Ask for disproof, not validation.
> Ask for evidence, not confidence.

In OFE, good engineers don't just use agents to produce. They use agents to argue, inspect, falsify, and verify.

PRINCIPLE 11

We Manage the Economics of Reasoning

OFE is also an economic model.

As software delivery becomes more agent-assisted, costs shift from people and licenses to tokens, context, retries, tool calls, orchestration, compute, storage, evaluation, and monitoring.

OFE Engineers should treat reasoning as a finite resource.

The question isn't just 'can the agent do it?' It's also 'was this the right amount of reasoning, context, model capability, and orchestration for the outcome?'

OFE Engineers know that waste now hides in prompts, context windows, repeated retries, unnecessary autonomy, and badly scoped work.

AI FinOps becomes part of engineering discipline.

PRINCIPLE 12

We Keep Humans in the Judgement Loop

Human-in-the-loop doesn't mean rubber-stamping agent output after the hard decisions are already made. It means humans keep judgment over intent, risk, ethics, architecture, financial exposure, production readiness, and organisational impact.

OFE Engineers define where autonomy helps and where judgment is mandatory.

> Automate execution where the contract is clear.
> Require human judgement where the consequence is material.

PRINCIPLE 13

We Build Systems That Can Be Operated, Not Demos That Can Be Admired

OFE Engineers don't stop at the prototype.

A prototype proves possibility. An OFE system proves reliability, operability, governance, and value.

That means every agentic or AI-assisted system needs:

Production concern	OFE expectation
Observability	Trace decisions, tool calls, failures and costs
Reliability	Define SLOs, retries, fallbacks and escalation
Governance	Maintain auditability and approval flows
Security	Control data, tools, permissions and identities
Drift	Monitor behavioural, data and cost drift
Supportability	Make failures understandable to humans
Improvement	Feed lessons back into rules, skills and contracts

OFE isn't 'AI helped us build it'. OFE is 'we can trust this to run'.

PRINCIPLE 14

We Continuously Compress Learning into the System

In Agentic Engineering we're always refining rules and skills iteratively, but too many eventually contradict each other or bloat context.

OFE Engineers need the same maintenance discipline.

After each delivery cycle, we ask:

> What rule should be added?
> What rule should be removed?
> What skill should be created?
> What skill should be retired?
> What context was missing?
> What context was unnecessary?
> What verification failed?
> What decision should become reusable?

A mature OFE practice gets lighter over time, not heavier.

COMMITMENT

HDS OFE Engineer's Oath

- We won't mistake activity for progress.
- We won't mistake code generation for engineering.
- We won't mistake autonomy for accountability.
- We won't mistake agent orchestration for architecture.
- We won't mistake a passing demo for a production system.
- We'll begin with the outcome.
- We'll define the contract.
- We'll curate the context.
- We'll separate research from implementation.
- We'll verify before we trust.
- We'll use agents to accelerate execution, inspection, and learning.
- We'll keep humans accountable for judgment, risk, and value.
- We'll build reusable skills, governed patterns, and evidence-based delivery loops.
- We'll optimise not only for speed, but for reliability, cost, explainability, and operational fit.
- We'll engineer forward from the outcome.

That's Outcome-Forward Engineering.

Outcome-Forward Engineers turn intent into measurable outcomes through disciplined context, governed execution, and evidence-based delivery.