

AEAF

AEAF: The Framework

The Agentic Enterprise Architecture Framework — for the human-agent enterprise

Author: Latentti Oy

Date: June 2026

About the author

Ari Hietamäki is an enterprise architect and AI practitioner based in Helsinki, with more than twenty-five years spent building and leading technology companies. He has founded and run businesses of his own, including a decade as chief executive, and through his consulting company **Latentti Oy** carries out enterprise-architecture work for Nordic organisations — the practice under which this framework is published. He now works at the intersection of enterprise architecture and applied AI: he is co-founder and CTO of an AI product company, and has served as an interim CTO leading the adoption of AI across the whole of an organisation's knowledge work — not as a pilot, but as the way the organisation runs. He is TOGAF®- and BPMN-certified, on top of a long line of infrastructure and security credentials earned across that career.

His work turns on the same boundary this book does: the line between deterministic work, which can be specified and tested, and non-deterministic agent work, which must be bounded and assured. He wrote *The AI-Driven Company* on the operating-model side of that question; AEAF is the architecture side. The observation in Chapter 3 — that an architecture principle has gained a second audience, the agents that now read and act on it at runtime — is his, and it is one of the seeds this framework grew from.

A note on how this book was made

This book was written the way it argues enterprises should now work: by a blended workforce. The author set the foundations — the thesis, the meta-model, the method, and a binding style guide — then allocated the drafting of each chapter to an AI agent, assigned the lens the chapter needed: enterprise-architecture or AI-architecture. Agents drafted; a craft pass and a cross-book quality review followed; the author reviewed every chapter and remained accountable for every claim, definition, and figure.

This is neither a novelty nor a thing to hide. It is the operating model the book describes, applied to the book itself. Each idea has a counterpart in how the text was produced: capability allocation (which chapters a human leads, which an agent drafts), oversight (a human **in-the-loop** on every chapter), guardrails (the style guide, enforced on human and agent authors alike), and the one rule the framework never relaxes — **accountability is never delegated to an agent**. The agents wrote a great deal of this text. Responsibility for it is the author's alone.

There would be something hollow in arguing for the blended workforce in a book produced without one. This book was built by the workforce it describes — it is the framework's own §19.7, *the architecture practice as a blended-workforce capability*, applied to its own making.

Contents

PART I · WHY ARCHITECTURE MUST CHANGE	11
Chapter 1 — The AI inflection for enterprise architecture	12
1.1 The moment: agents are doing real work	12
1.2 The central error: an agent is not "an app in the application layer"	13
1.3 Why this is an architecture problem, not a technology choice	14
1.4 What is at stake if architecture ignores it	15
1.5 What this book is, who it is for, and how to read it	17
1.6 What this book does not claim	19
Chapter 2 — What TOGAF gets right, and where it stops	20
In brief	20
2.1 What TOGAF is, and why it has endured	21
2.2 The assumption stack TOGAF was built on	23
2.3 The core appraisal: element → status → why → AEAf response	25
2.4 The shift, in one picture	34
2.5 Net assessment	35
Chapter 3 — The Agentic Enterprise	38
In brief	38
3.1 What enterprise architecture was built to do	39
3.2 The inflection: an agent is a new kind of actor	40
3.3 The four breaks	43
3.4 The new concern: trust, safety, and accountability as first-class architecture	48
3.5 The fused world: how work actually gets done	49
3.6 What this demands of enterprise architecture	53
3.7 AEAf's design tenets	55
PART II · FOUNDATIONS	58
Chapter 4 — Core concepts & definitions	59

In brief	59
1. The locked lexicon (master reference)	60
2. Actor, and the agent / model / application distinction	64
3. The autonomy spectrum	67
4. The determinism spectrum	71
5. Capability, capability allocation, and the automation frontier	72
6. The governance vocabulary	73
7. The two structural terms: the Intelligence & Agency domain and BDAT+I	77
8. What this chapter fixed, and where each concept is applied	78
Chapter 5 — The AEAf meta-model	80
In brief	80
5.1 What a content meta-model is, and why it is the backbone	81
5.2 The core extension at a glance	82
5.3 The full entity catalog	84
5.4 The three structural moves	92
5.5 The traceability the meta-model must preserve	95
5.6 How the meta-model is used	97
Chapter 6 — Architecture domains: BDAT + I	99
In brief	99
6.1 Why architecture is organized into domains	99
6.2 Why BDAT has no home for machine intelligence and agency	101
6.3 The fifth domain: Intelligence & Agency (the +I)	104
6.4 How the five domains relate	106
6.5 Where Data splits: transactional data and the cognitive substrate	108
6.6 The five domains at a glance	109
6.7 What this chapter fixes, and what comes next	111
Chapter 7 — The reworked Actor model & the blended workforce	112
In brief	112
1. The de-anthropomorphized Actor as the unit of organizational design	112
2. Capability allocation as a first-class decision	114

3. The automation frontier	118
4. Oversight modes	120
5. Hybrid teams	123
6. Worked example: the refund-settlement capability	124
7. What this chapter hands forward	127
Chapter 8 — Principles for the Agentic Enterprise	128
8.1 What a principle is for, and the structure AEAf keeps	128
8.2 The dual-audience principle	129
8.3 The Enforcement attribute	130
8.4 Compiling a principle into a guardrail	132
8.5 The AEAf Principle Catalog	135
8.6 How principles relate to the rest of AEAf	148
PART III · THE METHOD	150
Chapter 9 — The AEAf Method: overview & the cycle	151
9.1 Why a method, and why this one	151
9.2 The cycle	152
9.3 The centre: Intent & Assurance Management	155
9.4 The phases at a glance — the migration bridge	156
9.5 Phase 6 — Intelligence & Agency, the heart of the method	161
9.6 The dual governance loop	161
9.7 Tailoring the method	163
Chapter 10 — Vision	165
10.1 What the Vision phase is for	165
10.2 What is new in the agentic enterprise	166
10.3 Stakeholder management when stakeholders are affected by autonomy	168
10.4 Vision-phase inputs and outputs	170
10.5 What Vision assumes the Preliminary phase already produced	172
10.6 A worked vision vignette	173
Chapter 11 — Business & capability architecture	176
In brief	176

1. What Phase 2 produces, and what is new	176
2. Building the capability map with a performer dimension	178
3. The two signature outputs: allocation and the frontier map	179
4. Value-stream and process redesign for mixed execution	181
5. Organizational impact	183
6. Phase 2 inputs and outputs	184
7. Worked capability-map excerpt with allocations and a target frontier	186
8. What this chapter hands forward	188
Chapter 12 — Data & knowledge architecture	189
In brief	189
12.1 Classical data architecture does not retire	190
12.2 The new layer: the cognitive substrate	191
12.3 The Data ↔ Intelligence boundary	193
12.4 Architecture concerns unique to cognition	194
12.5 Provenance for traceability	198
12.6 Reference picture: data → corpus → retrieval → agent	198
12.7 Governing the cognitive substrate	200
12.8 Phase 3 inputs, outputs, and the Knowledge-Corpus Catalog	202
12.9 What Phase 3 hands on	203
Chapter 13 — Application architecture in a non-deterministic world	205
In brief	205
1. The assumption that breaks, and what it takes down with it	206
2. Designing for non-determinism	208
3. Where the agent runtime sits: component versus actor	212
4. Integration patterns updated for AI	214
5. Observability of probabilistic components	216
6. The application portfolio in the agentic enterprise	217
7. What this chapter fixes for the phases around it	220
Chapter 14 — Technology architecture	221
In brief	221

14.1 The classical technology layer still applies	222
14.2 The AI platform: the new technology stratum	223
14.3 Integration plumbing: MCP and tool gateways as standards	230
14.4 Cross-cutting technology concerns amplified by AI	231
14.5 The technology standards view (AI-stack Technology Portfolio)	233
14.6 A note on vendor surfaces (illustration only)	235
14.7 Phase 5 inputs and outputs	236
14.8 The Technology Radar	237
Chapter 15 — Intelligence & Agency architecture	242
15.1 What Phase 6 produces: the designed agentic workforce	242
15.2 The anatomy of an agent	244
15.3 The model layer and model selection as an architecture decision	247
15.4 How an agent draws on the cognitive substrate	249
15.5 Orchestration: single-agent, multi-agent, and human-in-the-loop	250
15.6 Tools and actions: the agent's ability to affect the world	253
15.7 Guardrails and autonomy by design	254
15.8 Evaluation by design	256
15.9 Phase 6 inputs and outputs	257
Chapter 16 — Opportunities, solutions & migration	261
16.1 From target to plan: what these two phases do	262
16.2 The new sequencing axis: trust earned over time	262
16.3 Phase 7 — Opportunities & Solutions in the Agentic Enterprise	264
16.4 Phase 8 — Migration Planning: sequencing, dependencies, risk	266
16.5 Planning frontier moves and the human transition	269
16.6 The business case in the Agentic Enterprise	271
Chapter 17 — Intent & Assurance management	274
17.1 Why requirements-at-the-centre breaks for probabilistic work	274
17.2 Intent & Assurance generalises requirements, it does not abolish them	275
17.3 Intent: a declared outcome with bounds, priority, and an acceptance signal	276
17.4 Guardrails: the bounds the work must not cross	278

17.5 Assurance: from one-time verification to continuous evaluation	279
17.6 The assurance case: a structured argument that the agent is fit to operate	281
17.7 How the centre feeds every phase and both loops	282
17.8 What this chapter establishes	285
PART IV · CROSS-CUTTING ARCHITECTURE	286
Chapter 18 — Trust, safety & accountability architecture	287
18.1 Why this is first-class architecture, not an audit afterthought	288
18.2 The accountability model: an agent acts, a human answers	289
18.3 The threat surface specific to agent actors	292
18.4 Defense in depth as an architecture pattern	297
18.5 Safety controls as architecture	300
18.6 Trust as earned and revocable	303
Chapter 19 — The human-agent operating model	307
In brief	307
1. The operating model: from allocation to a running way of working	307
2. Decision authority and the Human-Agent RACI	309
3. Escalation design: when and how an agent hands back to a human	312
4. Trust calibration and the human factors of teaming	317
5. The operating rhythm: reviews, audits, and feedback loops	320
6. What this chapter hands forward	322
7. The architecture practice as a blended-workforce capability	323
Chapter 20 — Continuous & runtime governance	327
20.1 Why gate-only governance is blind to a system that changes between meetings	328
20.2 Design-time governance: the gate, extended	329
20.3 Runtime governance: the loop	331
20.4 The agent-lifecycle state machine	335
20.5 What triggers a new design-time cycle	338
20.6 Incident response for autonomous action	339
20.7 Governance bodies and the evidence an Agentic Enterprise must keep	342
20.8 How this binds to Intent & Assurance, guardrails, and safety	344

20.9 What this chapter establishes	345
PART V · REALIZATION & REFERENCE	347
Chapter 21 — The AEAf artifact & deliverable catalog	348
21.1 The artifact taxonomy AEAf inherits	349
21.2 What changes in the classical artifacts	350
21.3 The nine new AEAf artifacts at a glance	352
21.4 The nine artifacts defined	355
21.5 How the artifacts trace to each other and to the meta-model	359
21.6 The artifact set as the Architecture Repository for an Agentic Enterprise	360
Chapter 22 — The AEAf maturity model	363
In brief	363
1. Why a maturity model, and why this one is different	363
2. The AEAf maturity levels	365
3. The eight maturity dimensions	368
4. How to assess: evidence per cell	372
5. Using maturity: where to start and how to set a target	374
Chapter 23 — Adopting AEAf alongside / migrating from TOGAF	376
23.1 The reassurance: this is extension, not replacement	377
23.2 The master correspondence: AEAf ↔ TOGAF	378
23.3 The migration path for a TOGAF shop	384
23.4 What to extend versus what to create new	387
23.5 Running AEAf and TOGAF side by side during transition	388
23.6 Common objections from a TOGAF practice, answered honestly	390
23.7 Where this chapter stops	391
Appendices	393
Appendix A — Glossary	393
Appendix B — TOGAF ↔ AEAf mapping	398
Appendix C — Reading paths	406
Appendix D — The AEAf principle catalog (index)	407
Appendix E — Meta-model quick reference	410

Part I • Why architecture must change

Chapter 1 — The AI inflection for enterprise architecture

In brief. AI agents have moved from demonstration to doing real work — booking travel, settling claims, writing and deploying code, moving money. The reflex of most organizations is to file each agent under the application layer, as one more system to run. That is the central error this book corrects. An agent is not an application; it is a new kind of **actor** — software that perceives, decides, and acts with goal-directed initiative, probabilistically, at machine speed. A new actor in the enterprise is an enterprise-architecture problem by definition: it cuts across business, data, application, technology, governance, and people at once, and no single one of those owns it. This chapter makes that case, names what is at stake when architecture ignores it, and sets out what the Agentic Enterprise Architecture Framework (AEAF) delivers across its two volumes and how to read them.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

1.1 The moment: agents are doing real work

Generative AI arrived first as a feature — a text box that answered questions, a copilot that suggested the next line of code. That phase is over. The systems now entering production are **AI agents**: a model placed in a loop with memory, tools, and an objective, able to take actions in the world toward a goal with limited or no step-by-step human direction. (We define the agent precisely, and distinguish it from the model that powers it, in → see AEAF Framework §4.) The shift that matters is not that the model got better. It is that the software now *acts*.

An agent that drafts an email is a feature. An agent that reads the inbox, decides which messages need a reply, sends those replies, and books the meetings they generate is something else. It has crossed from producing output a human then acts on, to taking the

action itself. Once that line is crossed, the questions stop being about model quality and start being about authority, oversight, and accountability — the questions architecture exists to answer.

The crossing is happening unevenly and quietly. A team stands up an agent to triage support tickets. A finance group lets one reconcile invoices. An engineering org gives one the ability to open and merge pull requests. Each is a local decision, made for a local reason, and none of them passes across an architect's desk. The enterprise acquires a population of acting software the way it once acquired spreadsheets — one pragmatic choice at a time, with no one holding the whole picture.

This is the inflection. Not a single dramatic launch, but a threshold being crossed in a hundred places at once: software has begun to do work that only people used to do, and the organization has no model for it.

1.2 The central error: an agent is not “an app in the application layer”

Faced with something new, an architecture practice reaches for the nearest existing box. An agent runs as software, calls APIs, and is deployed like a service — so it gets filed in the **application layer**, catalogued next to the CRM and the billing system. This is the error the whole book is built to correct.

An application, in the classical model, is a passive capability: it does what it is told, when it is told, and produces a defined output for a given input. It does not decide what to do next. It does not pursue a goal across steps. It does not act unless a human or a deterministic schedule invokes it. An agent does all of these things. Filing it as an application hides exactly the properties that make it consequential.

The honest comparison is not agent-versus-application. It is agent-versus-**actor** — the slot in the enterprise that, until now, only a human could occupy. An actor performs capabilities, holds responsibilities, consumes and produces information, and participates in processes. An agent does all of that, and differs from a human actor in ways that touch every architecture domain.

PROPERTY	HUMAN ACTOR	CLASSICAL APPLICATION	AI AGENT ACTOR
Initiative	Acts on goals and judgment	Acts only when invoked	Acts on a goal with limited direction

PROPERTY	HUMAN ACTOR	CLASSICAL APPLICATION	AI AGENT ACTOR
Behaviour	Variable, judgment-based	Deterministic, specifiable	Probabilistic — same prompt, different action
Speed & scale	Human pace, one task at a time	Bounded by its function	Machine speed; thousands of decisions/min, around the clock
Specification	Trained, instructed, supervised	Built to spec, tested to spec	Directed in natural language; partly learned disposition
Accountability	Personally answerable	The owning team is answerable	None intrinsic — a human or legal entity stays answerable

The table shows why "it's just an application" fails. An agent has the *initiative and behaviour* of an actor with the *deployment surface* of an application. Treat it as the latter and you govern the deployment while leaving the initiative ungoverned. The full argument — that this single mis-modelling breaks four foundational assumptions of classical EA at once — is the work of → see AEAF Framework §3.

1.3 Why this is an architecture problem, not a technology choice

A genuinely new kind of actor is not a procurement decision or an engineering detail. It is an architecture problem in the strict sense: it changes the relationships *between* the things an enterprise is made of, and it does so in more than one domain at the same time. No single team can absorb it, because no single team owns all the domains it touches.

Consider one agent — a procurement agent that can place orders up to a limit. It is not one team's concern; it lands in six at once.

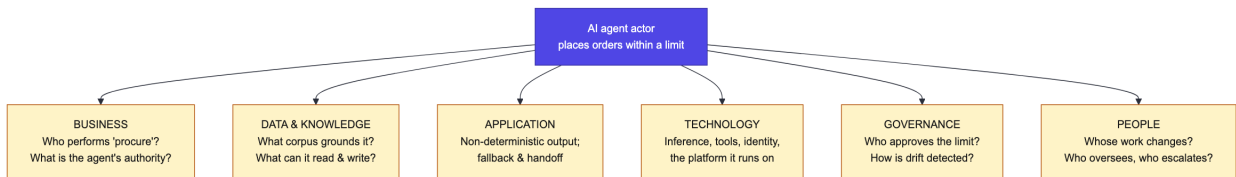


Figure 1.1 — One agent cuts across every domain at once. The indigo node is a single agent actor; the amber nodes are the six concerns it raises simultaneously. No one of them owns the agent, which is precisely why it is an enterprise-architecture problem rather than a single team's build decision.

Read Figure 1.1 as a list of unavoidable questions. The agent's **authority** is a business-architecture decision — which capability it performs and what it is permitted to do. The **knowledge** it draws on and the data it may read and write is a data-architecture decision. Its **non-deterministic output** — what happens when it is wrong, and where control hands back to a human — is an application-architecture decision. The **platform** that runs inference, manages its identity, and grants its tools is a technology decision. **Who approves its limit and how its behaviour is watched over time** is governance. And **whose work it changes, who oversees it, and who picks up an escalation** is a people decision.

Each question is real, each belongs to a different discipline, and they are coupled — change the authority and you change the oversight; change the knowledge and you change the failure modes. Coordinating coupled decisions across domains toward a coherent target state is the definition of enterprise architecture. That is why agents are architecture's problem, and why no narrower owner can hold it.

1.4 What is at stake if architecture ignores it

The cost of mis-filing agents is not abstract. When acting software is governed as if it were passive software, specific failures follow. They are already visible in organizations that moved fast without an architecture.

FAILURE MODE	WHAT IT IS	WHY FILING THE AGENT AS "AN APP" CAUSES IT
Ungoverned autonomy	Agents act at the edge of, or beyond, their intended authority — a refund agent issuing refunds it should have escalated.	Applications have fixed functions; no one set the agent's bounds, because applications don't need them.
Accountability gaps	When an agent's action causes harm, no named human is answerable — "the system did it."	Application ownership covers uptime, not decisions. No one was assigned to answer for what the agent chose.

FAILURE MODE	WHAT IT IS	WHY FILING THE AGENT AS "AN APP" CAUSES IT
Shadow agents	Agents stood up by teams outside any catalog or review, unknown to architecture, security, or risk.	With no actor model and no place to register an agent, the only path is the unsanctioned one.
Silent drift	An agent's behaviour changes — a new model version, a shifted prompt, a stale corpus — without anyone noticing it has stopped doing what it did.	Applications are tested once and assumed stable; probabilistic actors drift, and gate-only governance never looks again.
Untraceable decisions	A consequential action cannot be reconstructed: which agent, on what input, under what authority, grounded on what knowledge.	Application logs record calls, not the reasoning, authority, and oversight an actor's decision requires.

These are not edge cases waiting in the future. **Shadow agents** are the predictable result of having no place to put a sanctioned one — the same dynamic that produced shadow IT and unmanaged spreadsheets, now with software that acts. **Accountability gaps** are not a compliance footnote; when a regulator or a court asks who is answerable for an automated decision, "no one" is not an available answer. And **ungoverned autonomy** scales the way machines scale: a human making a bad judgment makes one; an agent making a bad judgment makes it a thousand times before lunch.

The deeper stake is institutional. An enterprise that cannot answer *who is permitted to do what, under whose accountability, watched how* — for its agents as it can for its people — does not actually know how it operates. Architecture's core promise is to keep that question answerable as the enterprise changes. Agents do not retire that promise; they raise its stakes.

1.5 What this book is, who it is for, and how to read it

AEAF is a TOGAF-class enterprise-architecture framework for the enterprise where humans and AI agents work as a blended workforce — the combined population of human and agent actors performing the enterprise's work across the same capabilities. It keeps the proven discipline of moving from a baseline to a governed target state, and rebuilds the parts of the classical model that a non-human actor breaks. It is delivered in two volumes.

VOLUME	TITLE	ANSWERS	THIS VOLUME
Book 1	AEAF — The Framework	What the agentic enterprise is and how to architect it: the actor model, the meta-model, the five domains, dual-audience principles, the method, continuous governance.	You are here.
Book 2	AEAF in Practice	How to adopt and run it: tailoring, playbooks, the new artifact templates, and a traceability spine back to Book 1.	→ see AEAF in Practice.

Who this book is for. Three readers, addressed without talking down to any:

- **The working enterprise or solution architect** who must put agents into a landscape and keep it coherent and governable.
- **The engineering, AI, or governance leader** accountable for systems that now act — and answerable for what they do.
- **An AI system itself**, consuming the architecture at runtime as a constraint it must obey, not only as documentation a person reads. (That an architecture now has a machine audience is itself one of the breaks AEAF answers; the dual-audience principle is developed in → see AEAF Framework §8.)

How Book 1 is organized. Five parts move from motivation to reference.

PART	CHAPTERS	WHAT IT ESTABLISHES
I — Why architecture must change	1–3	The inflection (this chapter), a fair TOGAF appraisal (Ch.2), and the founding thesis of the agentic enterprise (Ch.3).
II — Foundations	4–8	The locked definitions, the meta-model, the BDAT+I domains, the reworked actor model, and dual-audience principles.
III — The Method	9–17	The AEAF Method ** — its cycle and each phase, with Intent & Assurance at the centre in place of requirements-and-testing.
IV — Cross-cutting	18–20	Trust, safety & accountability; the human–agent operating model; continuous runtime governance.
V — Realization & reference	21–23, A	The artifact catalog, the maturity model, the migration bridge from TOGAF, and the appendices.

Reading paths. The book is a reference; few will read it cover to cover on the first pass.

- **Make the case to a board or a peer:** Chapters 1–3 stand alone as the argument for why this is architecture's problem.
- **Get the conceptual core fast:** Chapters 4–8 — definitions, meta-model, domains, actor model, principles.
- **Run an engagement:** Part III (the Method, Ch.9–17), with Part IV's governance chapters alongside.
- **Already practise TOGAF:** start with Chapter 2's appraisal and Chapter 23's migration bridge, then the foundations.

A note on what this chapter deliberately leaves to others. The **full thesis and the four broken assumptions** of classical EA are developed in → see AEAF Framework §3. The **fair appraisal of TOGAF** — what it gets right and exactly where it stops — is → see AEAF

Framework §2. The **locked definitions** of agent, model, autonomy, determinism, guardrail, intent, eval, and oversight are → see AEAF Framework §4. This chapter names them and points forward; it does not pre-empt them.

1.6 What this book does not claim

A framework earns trust partly by being clear about its limits. AEAF makes four disclaimers up front.

- **It is not the agentic TOGAF, and not a TOGAF profile.** AEAF answers and extends TOGAF where TOGAF bends, and replaces it only where it breaks. It is independent, original work, neither derived from nor endorsed by The Open Group. Where a parallel aids the reader, the book draws it fairly: where TOGAF uses the ADM, AEAF uses the AEAF Method. It never implies lineage, certification, or endorsement.
- **It does not predict the technology.** AEAF is an architecture framework, not a forecast of which models, vendors, or capabilities will lead. Its claims are built to survive the next model generation, because they concern how to *govern an acting actor*, not which actor is best this quarter.
- **It does not promise that agents are safe to deploy.** The framework gives you the means to *bound, oversee, and assure* agent behaviour and to keep a human accountable. It does not assert that any given agent is fit for any given task. That judgment stays with the accountability owner, by design.
- **It does not claim to be finished.** The agentic enterprise is being built while the framework is being written. AEAF is offered as the first disciplined, whole-enterprise treatment of the blended workforce — not the last word on it.

What AEAF does claim is narrow and falsifiable: that AI agents are a new kind of actor, that this is an enterprise-architecture problem, and that the discipline can be extended to design, allocate, govern, and assure a workforce that is part human and part machine. The rest of Book 1 makes that claim good.

Chapter 2 — What TOGAF gets right, and where it stops

In brief

TOGAF is the most complete enterprise-architecture method the discipline has produced, and AEAF keeps its spine. The governed move from baseline to target, run as an iterative cycle with requirements at the centre and governance as a standing discipline, is the right shape for any enterprise that intends to manage complexity on purpose rather than accrete it by accident. AEAF does not discard that machinery; it earns the right to extend it by first stating, accurately, what TOGAF does and why it has lasted. This chapter does three things. It names the eight-to-ten genuine strengths AEAF retains. It exposes the four-part assumption stack TOGAF was quietly built on — human actors, human readers, deterministic systems, gated change — and shows that the agentic enterprise makes all four false at once. And it renders, element by element, the appraisal at the core of AEAF: for each TOGAF construct, does it *hold*, *bend*, *break*, or is the concern simply *missing*? The net finding is that roughly three-quarters of TOGAF carries forward unchanged or extended; the deepest breaks cluster at Requirements, at Implementation Governance and Change Management (Phases G and H), and at the human Actor model; and four concerns have no TOGAF analogue at all, which AEAF must originate. The detailed responses are summarised here and developed in the chapters that follow.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. This chapter contrasts AEAF with TOGAF concepts fairly; it does not present AEAF as a TOGAF derivative, adaptation, profile, or certified extension.

2.1 What TOGAF is, and why it has endured

The Open Group Architecture Framework describes how an organisation's business, information, applications, and technology fit together today, designs where they should be, and governs the journey between the two. Its centrepiece is the Architecture Development Method (ADM): a phased, cyclic process that moves from vision through architecture and solutions into governance and change, and then loops. TOGAF has been the de-facto enterprise-architecture method for two decades because the things it formalised are the things enterprise architecture actually needs. They are not fashion; they are the load-bearing structure, and AEAf inherits them.

The endurance is not an accident of marketing. Each of the following constructs solves a problem that does not go away when the workforce changes.

The ADM as an iterative cycle. Vision → architecture → solutions → governance → change, looping rather than terminating, is the correct shape for an enterprise that is never "done." Architecture is a standing function, not a project that finishes. AEAf keeps the cycle and runs additional machinery *inside* it, not instead of it.

Requirements at the centre, touching every phase. TOGAF puts one persistent concern — Requirements Management — at the hub of the ADM, threading through every phase and keeping the architecture honest against what stakeholders actually need. The idea that a single discipline holds the whole cycle accountable is correct. AEAf keeps the centre and broadens what sits in it.

Capability-based planning. Planning by capability — a stable, outcome-named ability such as "Settle a claim" — rather than by project or by org chart, is one of TOGAF's strongest moves. It survives completely, and gains power: because a capability is defined independently of who performs it, *who or what performs it* (human, agent, or hybrid team) becomes a first-class allocation decision the model was already shaped to carry.

Governance discipline. Architecture boards, compliance reviews, exception management, decision rights recorded with rationale — the principle that architecture must be *governed*, not merely drawn, is more important in the agentic enterprise, not less. An enterprise that lets agents act without a governance function is not governing itself.

Stakeholder management. Power/interest analysis, tailored communication, and engagement planning all hold without modification. AEAf extends only the *set* of stakeholders, adding accountable owners for agent behaviour, not the method for managing them.

The Architecture Repository. A governed, versioned, discoverable store of architecture assets — principles, standards, reference models, building blocks, contracts, the governance log — is exactly what a complex enterprise needs to avoid re-deciding the same thing twice. AEAF keeps the repository intact.

Principle structure (Name / Statement / Rationale / Implications). TOGAF's four-part principle shape is clean, durable, and widely adopted. It survives. AEAF adds one attribute to it rather than redesigning it.

Building blocks (ABBs and SBBs). Reusable, composable Architecture Building Blocks and Solution Building Blocks remain the right unit of reuse. AEAF introduces new *types* of building block — agents, models, guardrails — rather than abandoning the concept.

Baseline → Target → Gap → Roadmap. The core analytical move of all architecture work — describe what is, design what should be, name the gap, sequence the road — is untouched. AEAF performs it across one extra domain and a richer meta-model, but the move itself is sound.

Architecture contracts. Formal, accountable agreements on what will be delivered and to what standard remain essential. They become, in AEAF, the natural home for autonomy limits and oversight obligations — an extension of an existing instrument, not a new one.

Table 2.1 summarises the retained spine.

Table 2.1 — TOGAF constructs AEAF retains as its spine

TOGAF CONSTRUCT	WHAT IT DOES	WHY AEAF KEEPS IT
ADM iterative cycle	Governed, looping baseline→target development	Architecture is never "done"; the cycle is the right shape
Requirements at the centre	One persistent concern threads all phases	A single discipline that holds the cycle accountable is correct
Capability-based planning	Plan by outcome-named ability, not by project	Actor-agnostic capabilities let work be re-allocated human↔agent
Governance discipline	Boards, reviews, exceptions, decision rights	More necessary when non- human actors can act, not less
Stakeholder management	Power/interest analysis, tailored engagement	Method holds; only the stakeholder set grows

TOGAF CONSTRUCT	WHAT IT DOES	WHY AEAf KEEPS IT
Architecture Repository	Governed, versioned store of architecture assets	Avoids re-deciding; agents need parts of it too
Principle structure	Name / Statement / Rationale / Implications	Clean and durable; AEAf adds one attribute
Building blocks (ABB/SBB)	Reusable units of architecture and solution	Agents, models, guardrails become new block types
Baseline→Target→Gap→Roadmap	The core analytical move	Sound; performed across a fifth domain
Architecture contracts	Accountable delivery-and-standard agreements	Natural home for autonomy limits and oversight duties

This is not a courtesy list. It is the reason AEAf is an *extension* and not a replacement: remove these constructs and there is no enterprise architecture left to do. AEAf's task is to identify, precisely, the smaller set of places where the proven machinery meets an assumption that no longer holds.

2.2 The assumption stack TOGAF was built on

TOGAF works because its constructs were correct for the enterprise it was designed to serve. That enterprise rested on four assumptions so obviously true at the time that they were never written down. The agentic enterprise — one staffed by a blended workforce of humans and AI agents — makes all four false at once. Naming them is the hinge of this chapter: every place TOGAF bends or breaks traces back to one of these four.

Table 2.2 — The four assumptions baked into classical enterprise architecture

#	ASSUMPTION	HELD BECAUSE	WHAT THE AGENTIC ENTERPRISE DOES TO IT
A1	Actors are human. Work is performed by people and the deterministic systems they operate.	For the whole history of the discipline, only humans exercised judgment and initiative.	An AI agent decides and acts. The workforce is now part human, part agent, part hybrid team.

#	ASSUMPTION	HELD BECAUSE	WHAT THE AGENTIC ENTERPRISE DOES TO IT
A2	Architecture is read by humans. Principles, standards, and models are documentation that people interpret and apply.	The only consumer of an architecture artifact was a person.	Agents read architecture at runtime. A principle becomes a constraint an agent must obey while acting.
A3	Systems are deterministic and specifiable. Given an input, a correct system produces a defined output; behaviour can be specified up front and verified by test.	Software was rule-based; correctness was conformance to spec.	An agent is probabilistic. The same prompt can yield different actions. "Correct" gives way to "acceptable within bounds."
A4	Change is episodic and gated. Architecture changes through deliberate, governed cycles; between cycles the landscape is stable.	Systems did not rewrite their own behaviour or learn in production.	Agents act, learn, and drift at machine speed, between governance meetings, with no change request.

An AI agent is not a new application to slot into the technology layer. It is a model placed in a loop with memory, tools, and an objective, able to take consequential action toward a goal with limited or no step-by-step human direction. It therefore occupies the conceptual slot that, until now, only a human could occupy: an *actor* that performs capabilities, holds responsibilities, consumes and produces information, and participates in processes. Treating such an actor as merely a box in the application layer is the central modelling error AEAf exists to correct, and it is the modelling error that A1 makes structurally invisible.

Two clarifications keep the contrast fair. First, the assumptions were *good* assumptions — they were true, and building on them was correct. TOGAF is not wrong about the enterprise it was built for. Second, the breaks are not everywhere. Most of TOGAF never touches these four assumptions in a way that matters; the determinism assumption, for instance, is

irrelevant to stakeholder management. The discipline of the next section is to find exactly where an assumption is load-bearing for a construct, and to grade the consequence honestly.

2.3 The core appraisal: element → status → why → AEAF response

This is the heart of the chapter. For each TOGAF element, AEAF assigns one of four statuses, then states why and what AEAF does in response. The status legend is fixed and used identically throughout both books.

Status legend. *Holds* — works unchanged. *Bends* — sound but needs extension. *Breaks* — a core assumption fails; the element needs replacement at its centre. *Missing* — the concern has no TOGAF analogue and must be originated.

The AEAF responses below are deliberately summarised. Each is developed in full later in Book 1; the forward references point to where. The four *Missing* rows are the genuinely new work and are kept prominent.

Table 2.3 — TOGAF element → AI-age status → why → AEAF response

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Preliminary Phase	Bends	The architecture capability must stand up agent-relevant assets before any cycle: model inventory, guardrail catalogue, eval infrastructure, an autonomy taxonomy; principles must be authored to be machine-consumable, not only readable.	Establish the Intelligence & Agency capability and a dual-audience principle authoring standard up front. → Ch.6, Ch.8

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Phase A — Vision	Holds	Scoping, stakeholder analysis, vision, and Statement of Work remain valid framing.	Keep. Add a blended-workforce intent to scope: which outcomes are human, agent, or hybrid by design, and the target autonomy posture. → Ch.10
Phase B — Business Architecture	Bends	Capabilities, value streams, and processes hold, but the performer is no longer assumed human; allocation (human/agent/hybrid) becomes a primary design variable and value streams gain escalation steps.	Add a Capability-Allocation model and human-in/on-the-loop checkpoints in value streams. → Ch.11
Phase C — Data Architecture	Bends	Conceptual/logical/physical models hold, but agents consume data as knowledge corpora, embeddings, retrieval contexts, and lineage that must be machine-traceable for accountability — data is now grounding material, not only record.	Model the Knowledge Corpus, grounding sources, and lineage-for-accountability. → Ch.12

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Phase C — Application Architecture	Bends	Components, interfaces, and integration hold, but a new class of component — the agent — is an active, non-deterministic participant that calls models, invokes tools, and orchestrates other agents. Static interaction matrices understate runtime behaviour.	Make Agent and Model first-class application-layer component types; model tool access and agent-to-agent orchestration. → Ch.13
Phase D — Technology Architecture	Bends	Compute, network, storage, and security still apply, but a new stratum appears: inference/model-serving, vector stores, eval/observability pipelines, guardrail enforcement points; security must cover prompt injection, model exfiltration, and agent identity.	Add the agentic technology stratum and agent-aware security (agent identity, action authorization, runtime policy points). → Ch.14
Phase E — Opportunities & Solutions	Holds	Work-package definition, transition architectures, and roadmapping remain valid delivery machinery.	Keep. Recognise agent rollout, eval-suite build, and autonomy-graduation as work-package types. → Ch.16

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Phase F — Migration Planning	Bends	Sequencing and dependency logic hold, but migration now graduates agents through autonomy levels with assurance evidence gating each step — not a one-time cutover.	Add autonomy-graduation gates: each increase in authority requires accumulated eval evidence and an updated accountability owner. → Ch.16
Phase G — Implementation Governance	Breaks	Governance assumes change is episodic and reviewed before go-live. Agents act and adapt continuously, post-deployment, faster than any board can convene; design-time review cannot see behaviour that only exists at runtime.	Dual governance: keep design-time gates and add runtime policy enforcement plus continuous assurance, with automated escalation. → Ch.20
Phase H — Change Management	Breaks	Change is treated as monitored, assessed, and fed into a future cycle. Agentic systems change their own effective behaviour (learning, model swaps, context drift) between cycles, with no human change request.	Replace episodic handling with continuous assurance: re-evaluate on cadence and on trigger; material behavioural change auto-opens a governance event. → Ch.20

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Requirements Management (centre)	Breaks	Requirements assume behaviour can be specified and tested to pass/fail. Probabilistic systems cannot be exhaustively specified; correctness is distributional, not binary.	Replace pure Requirements with Intent & Assurance at the centre: declare intended behaviour and bounds, then evaluate against eval suites and thresholds. Deterministic requirements still apply unchanged. → Ch.17
The BDAT four-domain model as complete	Breaks	BDAT has no home for agency, cognition, autonomy, or the model/knowledge/eval assets that now drive behaviour; spreading them across BDAT loses them as a coherent concern.	Add a fifth domain — I, Intelligence & Agency . BDAT → BDAT+I . → Ch.6
Content meta-model — Actor	Breaks	Actor is implicitly human; modelling an agent as an “application” loses its decision-making nature.	Introduce a human / agent / hybrid actor model. Agent is a distinct actor type with autonomy level, oversight mode, and a named accountable human owner. → Ch.5, Ch.7
Content meta-model — Role	Bends	Roles still describe responsibility, but a role may be filled by a human, an agent, or a pairing, with different accountability semantics.	Add an occupant type (human/agent/hybrid) and link agent-occupied roles to an Accountability Owner. → Ch.5

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Content meta-model — Function / Process	Bends	Functions and processes hold, but steps may be agent-executed and probabilistic, requiring oversight and escalation, not only control flow.	Annotate steps with execution mode, autonomy level, and oversight/escalation points. → Ch.5, Ch.11
Content meta-model — Application Component	Bends	Valid for deterministic services; insufficient for agents and models, which behave non-deterministically and consume policy at runtime.	Keep the component; add Agent and Model as related-but-distinct entity types with their own attributes. → Ch.5
Content meta-model — Data Entity	Bends	Holds for structured transactional data; underspecified for unstructured corpora, embeddings, and grounding context.	Add Knowledge Corpus as an entity type with provenance, eligibility, and lineage. → Ch.5, Ch.12
Content meta-model — Technology Component	Holds	Still the right entity for infrastructure.	Keep; model inference, vector store, and eval pipeline as Technology Components with agent-relevant attributes. → Ch.14

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Architecture Principles	Bends	The four-part structure holds, but principles now have a second audience: agents consume them at runtime as constraints. Prose alone cannot be enforced on an agent.	Add an Enforcement attribute — advisory / design-time / runtime guardrail — with a machine-consumable expression for runtime principles. → Ch.8
Architecture Governance (the function)	Breaks	Governance designed for human-paced, pre-deployment control cannot govern systems that act and adapt continuously after deployment.	Dual governance as the standing model: design-time board governance plus runtime enforcement and continuous assurance. → Ch.20
Architecture Board	Bends	The board remains the right human decision-and-dispute body, but cannot be the runtime control loop and lacks agent-behaviour expertise.	Keep for strategy, autonomy-graduation, and accountability assignment; add Trust & Safety competence; feed it continuous-assurance signals. → Ch.18, Ch.20
Architecture Contracts	Bends	Contracts hold, but did not contemplate agent authority, autonomy limits, or oversight duties.	Extend the contract to specify autonomy level, guardrails in force, oversight obligations, and the named Accountability Owner. → Ch.18

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Compliance Reviews	Bends	Point-in-time, pre-go-live reviews remain useful for design conformance but cannot assess behaviour that only emerges at runtime.	Keep design-time review; pair it with continuous compliance against guardrails and eval baselines. → Ch.20
Artifacts — Catalogs	Bends	Catalogs hold but are incomplete without the new entities.	Add Agent Catalog, Model Portfolio, Guardrail/Policy Catalog, Eval-Suite Specification, Knowledge-Corpus Catalog. → Ch.21
Artifacts — Matrices	Bends	Matrices hold but must now express who or what performs and how autonomously.	Add the Capability-Allocation matrix and an agent-to-model / agent-to-tool access matrix. → Ch.21
Artifacts — Diagrams	Bends	Diagrams hold but understate runtime, non-deterministic, escalating behaviour.	Add an oversight/escalation flow view and an autonomy-boundary view. → Ch.21
Architecture Repository	Bends	Holds as a human-facing governed store, but agents now need parts of it (principles, standards, guardrails) machine-readable at runtime.	Add a machine-consumable runtime layer, versioned alongside the human-readable assets. → Ch.20, Ch.21

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Standards Information Base (SIB)	Bends	Still the right place for approved standards, but must now hold model, eval, and guardrail standards.	Extend the SIB with approved models, eval-coverage standards, autonomy-level definitions, and guardrail standards. → Ch.21
TIME portfolio model	Bends	TIME still rationalises a portfolio, but has no axis for agentic potential.	Add an agentic-disposition axis (Augment / Agentify / Keep-Human / Retire) alongside TIME. → Ch.16
Skills Framework / architecture roles	Bends	The competency model holds, but the architect's remit now includes the blended workforce, autonomy posture, and assurance — and no role is accountable for agent behaviour.	Add Intelligence & Agency literacy across all architects, explicit Accountability Ownership, and a Trust & Safety strand. → Ch.18
Stakeholder Management	Holds	Power/interest analysis, tailored communication, and engagement all remain valid.	Keep. Extend the stakeholder set to include Accountability Owners and the Trust & Safety function. → Ch.18
Agent-action accountability & RACI	Missing	TOGAF has RACI for architecture work, but nothing for who is accountable when an autonomous agent acts in production and causes an outcome.	Originate an Accountability Owner entity and an agent-action RACI: a named accountable human, defined escalation, traceable decision lineage. → Ch.18, Ch.19

TOGAF ELEMENT	STATUS	WHY IT CHANGES IN THE AGENTIC ENTERPRISE	AEAF RESPONSE (DETAILED → CHAPTER)
Human-agent collaboration & escalation	Missing	TOGAF assumes humans do the work; it has no model for how humans and agents divide labour, hand off, supervise, and escalate.	Originate the human-agent operating model: in/on/out-of-the-loop oversight, handoff and escalation patterns. → Ch.7, Ch.19
Trust, Safety & Accountability (cross-cutting)	Missing	TOGAF treats security as a technology concern; it has no cross-cutting concern for agent safety, behavioural risk, harm prevention, and accountable autonomy.	Originate Trust, Safety & Accountability as a first-class cross-cutting concern spanning all five domains. → Ch.18
Evaluation-instead-of-testing assurance	Missing	TOGAF's verification assumes test-to-spec pass/fail; probabilistic behaviour has no such clean test.	Originate the Eval Suite as a first-class artifact and "evaluate against bounds and thresholds" as the assurance method. → Ch.17

Read the table by status, not row by row. The *Holds* rows confirm the spine. The *Bends* rows — the large majority — are where AEAF adds attributes, entities, and views to constructs that keep their shape. The five *Breaks* rows are where a core assumption fails at the centre of an element. The four *Missing* rows are the genuinely new architecture AEAF must originate, and they are where the framework spends most of its inventive effort.

2.4 The shift, in one picture

The appraisal reduces to a small number of structural moves. Figure 2.1 shows the TOGAF spine AEAF keeps, and the five places where it extends or replaces. Per the AEAF visual convention, indigo denotes machine/agent intelligence and amber denotes human

judgment; here the convention is applied at the level of the framework moves themselves — the kept spine is neutral, the agentic additions are indigo, and the additions that re-introduce human judgment (Intent, accountability) carry amber.

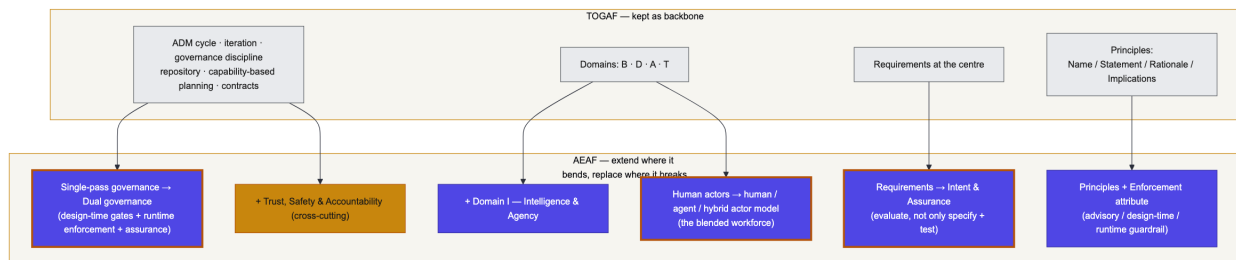


Figure 2.1 — The TOGAF→AEAf shift. The neutral nodes are the TOGAF spine AEAf retains unchanged or lightly extended. The six AEAf nodes are the structural moves: one new domain (+I), one new principle attribute (Enforcement), and four answers to broken assumptions — Intent & Assurance for the centre, dual governance for the ADM's governance phases, the reworked actor model for BDAT's human bias, and a cross-cutting Trust, Safety & Accountability concern. The amber and split-colour nodes mark where human judgment is deliberately re-introduced — Intent, accountability, and the hybrid actor model — so that adding machine intelligence does not remove the answerable human.

2.5 Net assessment

Most of TOGAF survives structurally. Of the elements appraised in Table 2.3, the large majority *hold* or *bend*. The method, the cycle, the four classical domains, the repository, governance as a discipline, capability-based planning, contracts, stakeholder management, and the principle structure all remain. Roughly three-quarters of TOGAF carries forward either unchanged or extended. The framework is being answered and built upon, not replaced — and that is the point. AEAf earns the right to add what it adds precisely because it leaves the proven structure standing.

The deepest breaks cluster at three points, and they share one root cause: TOGAF assumes deterministic systems that humans operate on episodic, gated cycles.

- 1. Requirements → Intent & Assurance.** You cannot specify-and-test probabilistic behaviour to a clean pass/fail. The centre of the method has to shift from specification to bounded behaviour evaluated against eval suites. (→ see AEAf Framework §17.)
- 2. Governance G and Change H → dual governance plus continuous assurance.** Pre-deployment, human-paced review cannot govern systems that act and adapt

continuously after go-live. Design-time gates must be paired with runtime enforcement and ongoing assurance. (→ see AEA Framework §20.)

3. The four-domain model and the human Actor. BDAT has no home for agency, and the meta-model's Actor is implicitly human. Both break against a workforce that is part non-human and decision-making. (→ see AEA Framework §5, §6.)

The genuinely new things AEA must originate — the four concerns with no TOGAF analogue at all — are where the framework does its most original work.

Table 2.4 — What AEA must originate (the four Missing concerns)

NEW CONCERN	WHY TOGAF HAS NO ANALOGUE	WHERE AEA BUILDS IT
Human-agent collaboration & escalation model	TOGAF assumes humans do the work; no model for dividing labour, handoff, supervision, escalation	→ see AEA Framework §7, §19
Accountability & RACI for agent actions	RACI exists for architecture work, not for who is answerable when an autonomous agent acts in production	→ see AEA Framework §18, §19
Evaluation-instead-of-testing assurance	Verification assumes test-to-spec pass/fail; probabilistic behaviour has no clean test	→ see AEA Framework §17
Intelligence & Agency domain (+I) and Trust, Safety & Accountability	BDAT has no coherent home for agents, models, autonomy, knowledge, and evals; security is treated as technology, not a behaviour-focused cross-cutting concern	→ see AEA Framework §6, §18

Everything else is extension, not invention — and that balance is deliberate. A framework that replaced TOGAF wholesale would ask the discipline to throw away two decades of working practice to absorb a workforce change that touches, sharply, only a handful of constructs. A framework that ignored the breaks would leave the agentic enterprise architected by a method blind to the actor that now drives its behaviour. AEA takes the third path: keep the spine, name the four assumptions that no longer hold, and extend or replace only where one of them is load-bearing. The chapters that follow build each response in turn — the meta-model and the de-anthropomorphised Actor (→ see AEA Framework §5), the five domains (→ see AEA Framework §6), the reworked actor model

and blended workforce (→ see AEAF Framework §7), dual-audience principles and the Enforcement attribute (→ see AEAF Framework §8), the AEAF Method and Intent & Assurance at its centre (→ see AEAF Framework §9, §17), and continuous, runtime governance (→ see AEAF Framework §20).

Chapter 3 — The Agentic Enterprise

In brief

Enterprise architecture was built for an enterprise made of people and deterministic machines. That premise held for the entire history of the discipline, and it was so obviously true that no framework ever wrote it down. Generative AI, and above all the **AI agent** — a model placed in a loop with memory, tools, and an objective — introduces a third kind of participant: software that perceives, decides, and acts with goal-directed initiative, probabilistically, at machine speed. This is not a new application to file under the technology domain. It is a new kind of **actor** in the organization, and its arrival breaks four foundational assumptions that frameworks such as TOGAF were built on. This chapter states those four breaks, names the one genuinely new concern they create — trust, safety, and accountability — and describes the operating picture that results: the **blended workforce**, in which a single capability may be performed by a human, by an agent, or by both, and in which that allocation moves over time. The chapter closes with what this demands of enterprise architecture and the design tenets AEAF answers it with. The formal entity model is developed in Chapter 5, the reworked actor and operating model in Chapter 7, and the method in Chapter 9; the thesis is stated in full here.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where this chapter contrasts the two, it does so to mark a real difference in assumptions, not to claim lineage.

3.1 What enterprise architecture was built to do

Enterprise architecture exists to connect strategy to execution. It describes how an organization's business, information, applications, and technology fit together today; it designs where they should be; and it governs the journey between the two. The discipline's central achievement was to make that journey *governed* rather than accidental — to replace ad-hoc change with a deliberate, traceable progression from a baseline to a target.

TOGAF formalized this as the Architecture Development Method (ADM): a cyclic, governed progression through architecture vision, the business / data / application / technology domains, opportunities and migration planning, implementation governance, and change management — with a requirements process at its centre feeding every phase. The ADM is good engineering. It encodes hard-won discipline: state the baseline, design the target, analyse the gap, sequence the work, govern the change. AEAF keeps that spine. The argument of this chapter is not that the method is wrong. It is that the method rests on assumptions that the agentic enterprise quietly invalidates, and that a method cannot be sound when its premises have changed underneath it.

The assumption stack classical EA was built on

Four assumptions sat beneath the whole discipline. None was ever stated as a premise, because each was simply the world as it was.

#	ASSUMPTION	WHY IT HELD, UNEXAMINED
A1	Actors are human. The enterprise's work is performed by people, and by the deterministic systems those people operate.	For the entire history of the discipline, only humans exercised judgment and initiative. Software executed; it did not decide.
A2	Architecture is read by humans. Principles, standards, and reference models are documents that architects and engineers interpret and apply.	The only consumer of an architecture artifact was a person. A principle was advice to a designer, never an instruction to a running system.

#	ASSUMPTION	WHY IT HELD, UNEXAMINED
A3	Systems are deterministic and specifiable. Given an input, a correctly built system produces a defined output; behaviour is specified up front and verified by testing.	Software was rule-based. Correctness meant conformance to a specification, and conformance could be proven by test.
A4	Change is episodic and gated. The architecture changes through deliberate, governed cycles; between cycles the landscape is stable.	Systems did not rewrite their own behaviour or learn in production. What you deployed on Friday behaved the same on Monday.

These four are not peripheral. They are load-bearing. The ADM's requirements-centred design assumes A3. Its architecture-board gating assumes A4. Its actor and role catalogues assume A1. Its principle catalogue — guidance written for a human to read and apply — assumes A2. Remove any one and a part of the method loses its footing. The agentic enterprise removes all four at once.

3.2 The inflection: an agent is a new kind of actor

The disruption is widely mis-located. It is tempting to say "the disruption is AI," or "the disruption is the large language model." Neither is right, and the distinction matters for everything that follows. To see why, three things that are routinely collapsed into one have to be held apart.

Model, application, and agent are three different things

CONCEPT	WHAT IT IS	WHAT IT IS NOT
Model	The underlying machine-learning model — typically a large language or multimodal model — that provides reasoning and generation. A powerful but passive function: text or data in, text or data out. One model can power many agents.	Not an actor. A model takes no initiative; it answers when called. It holds no objective, no memory across calls, no ability to act in the world.
Application	A software component with a defined interface and behaviour, possibly one that calls a model — a translation feature, a summarisation endpoint, a classifier in a pipeline. It does a bounded, specified job.	Not an actor either, even when it embeds a model. It executes a function; it does not pursue a goal across steps or choose its own actions.
AI agent	A model placed in a loop with memory , tools , and an objective — able to perceive a situation, decide what to do, take actions in the world (call systems, move money, send messages, write code, make decisions), observe the result, and continue toward the goal, with limited or no step-by-step human direction.	Not “a chatbot,” not “a model with a nicer interface,” not “an application.” The loop, the tools, and the objective are exactly what make it act rather than merely respond.

A generative model on its own is a function you invoke. An application that wraps it is still a tool you operate. The agent is the thing that changes the picture, because the agent **occupies a slot in the enterprise that until now only a human could occupy**: an entity that performs capabilities, holds responsibilities, consumes and produces information, takes consequential action, and participates in processes on its own initiative. That is the definition of an actor. The agent is a non-human actor.

What makes an agent an actor — and an actor unlike any before it

An agent is an actor because it acts toward a goal with initiative. But it is an actor with a profile no human actor and no classical system ever had, and each of these properties has architectural consequences.

DEFINING PROPERTY	WHAT IT MEANS	WHY IT MATTERS TO ARCHITECTURE
Probabilistic	The same prompt can yield different actions on different runs. Behaviour is sampled, not computed.	You cannot specify it exhaustively or prove it correct by test. Assurance must replace verification. → breaks A3.
Machine-speed and machine-scale	Thousands of decisions a minute, around the clock, in parallel, without fatigue.	A quarterly review cannot govern a system that takes a million actions between meetings. Governance must reach into runtime. → breaks A4.
Natural-language-directable	Its "specification" is partly an instruction you write and partly a disposition it learned in training. You direct it in prose, not in code.	Its directing instructions are now architecture artifacts an agent must read — and the prose itself is a control surface. → breaks A2.
Tireless and literal	It pursues the objective it was given, relentlessly, including in ways its designer did not intend or foresee.	Intent must be bounded explicitly; an under-specified goal pursued literally at scale is a hazard, not a convenience.
No intrinsic accountability	When an agent acts, no agent is answerable. A human or legal entity remains accountable for what it does.	Accountability cannot be delegated to the actor that performs the work — a structural break from every prior actor model, where performer and answerable party were the same person.

The single most consequential modelling error in the agentic enterprise is to treat such an actor as "an application" — a box in the application domain, governed like any other component. An application does a specified job and is verified once. This actor decides,

acts, drifts, and must be governed while it runs. Filing it under Applications hides exactly the properties that make it dangerous and valuable. AEAF exists, in large part, to correct that one error and to carry its consequences through every domain.

3.3 The four breaks

The assumption stack does not bend gently as agents arrive. It breaks at four named points. Each break is stated below with what fails, why it fails, and how AEAF answers it at the framework level. The full table follows the four subsections.

Break 1 — A1 fails: the workforce is no longer only human

What breaks. Agents decide and act. The "workforce" is therefore no longer the population of people the organization employs and the deterministic tools they operate. It is now **part human, part agent, and part hybrid team** — humans and agents performing a single capability together, with shared context and handoffs between them. The organization chart, the role catalogue, and the capability map that quietly assumed a human performer no longer describe the enterprise. A capability such as "triage an incoming claim" may today be performed by a person, tomorrow by an agent within bounds, and in between by the two in concert.

Why it breaks. A1 held because only humans exercised judgment and initiative. An agent exercises both — imperfectly, probabilistically, but genuinely enough that the work gets done without a human performing it. Once a non-human entity performs a capability, every model that equated "performer" with "person" is describing only part of the workforce.

AEAF's answer. A **reworked actor model** runs through every domain. **Actor** becomes an abstract type that specializes into **human actor**, **agent actor**, and **hybrid team**. The unit of design is no longer the org chart but the **blended workforce** — the combined population of human and agent actors allocated across the same capabilities. Capability allocation — *which kind of actor performs this, at what autonomy level, under what oversight* — becomes a first-class architecture decision rather than an implementation detail. → see AEAF Framework Ch.7 for the actor model and operating model; Ch.5 for the entity definitions.

Break 2 — A2 fails: architecture now has a second audience that reads it at runtime

What breaks. This is the break that is easiest to miss and most far-reaching once seen. Architecture artifacts — principles, standards, policies — were written for one audience: a human who reads them and applies judgment. A principle such as *"customer personal data is never sent to a third-party service"* was guidance to an engineer, advisory in force, applied at design time by a person who understood the intent.

In the agentic enterprise, an agent reads and is bound by that same principle **while it acts**. When an agent is deciding whether to call an external enrichment API with a customer record in hand, the principle is no longer advice to a distant designer. It is a constraint the agent must obey at the moment of action — or violate at machine speed, a thousand times, before any human notices.

This is Ari's observation, and it is the hinge of the whole framework: **architecture principles have gained a new audience. They are no longer only read by people. They are read — and must be obeyed — by agents, at runtime.** A discipline whose normative artifacts were all written for human interpretation now has machine readers in the loop, and those readers act on what they read without the judgment a human brings to a borderline case.

Why it breaks. A2 held because the only consumer of an artifact was a person, and a person reconciles a terse principle with context, precedent, and common sense. An agent does none of that unless the artifact is expressed in a form it can enforce. Prose that a human reads as "obviously, don't do the harmful version of this" is, to an agent, either silent or ambiguous. The artifact has to carry its meaning in a form the new audience can act on.

AEAF's answer. Dual-audience artifacts. Every normative artifact is written for humans *and* expressible for machines. Concretely, a **Principle** gains an **Enforcement** attribute with three values:

ENFORCEMENT	MEANING	AUDIENCE AND MECHANISM
Advisory	Guidance only, as in classical EA.	Human reader; applied by judgment.
Design-time	Checked at an architecture gate before deployment.	Human or automated check at a gate.
Runtime	Enforced continuously on agent behaviour as it acts.	Machine reader; compiled into one or more guardrails that bound the agent at the moment of action.

A principle marked `runtime` is **compiled into a guardrail** — a machine-enforceable constraint (an input filter, an output check, a tool permission, a spend limit, an action veto) that holds regardless of the agent's reasoning. The principle is what the enterprise believes; the guardrail is what the agent is actually constrained by. The dual-audience property is not decoration. It is the formal answer to "architecture now has a second reader." → see AEAF Framework Ch.8 for the principle catalogue and the Enforcement attribute; Ch.5 for the Principle→Guardrail compilation.

Break 3 — A3 fails: you cannot specify-and-test a probabilistic actor

What breaks. Classical assurance is *specify, build, verify*: write the requirement, build to it, test that the output matches, sign off. This works because deterministic systems are specifiable — there is a defined right answer and a test that detects deviation from it. A probabilistic actor has no such property. You cannot enumerate every situation it will meet, cannot specify the single correct action for each, and cannot prove by test that it will always take it. "Correct" gives way to **"acceptable within bounds, most of the time, with known failure modes."** Requirements-and-testing as the assurance model does not bend here; it collapses, because its central premise — a specifiable right answer verifiable once — is gone.

Why it breaks. A3 held because software was rule-based and conformance to spec was both definable and provable. An agent's behaviour is sampled from a distribution shaped by training and instruction. There is no specification to conform to, only a distribution to bound and observe. A test that passes on Tuesday says little about Thursday, and nothing about an input the test set never contained.

AEAF's answer. **Intent & Assurance** replaces pure requirements at the centre of the method. Instead of specifying the output, you:

1. **Declare intent** — the outcome the actor is to achieve and the bounds within which it must stay ("resolve the refund within policy; never exceed EUR 500 without human approval; never disclose another customer's data").
2. **Set guardrails** — the machine-enforceable limits that hold even when the agent's reasoning is wrong.
3. **Evaluate continuously** — run an **eval suite** (a maintained set of cases measuring quality, safety, alignment, and cost) on a cadence, and watch live behaviour, rather than verifying once and signing off.

Verification becomes **assurance**: the continuing evidence, from evals and guardrail logs and oversight, that the agent stays within its intent and bounds over time. Assurance is a discipline that runs as long as the agent runs, not an event that happens before launch. → see AEAF Framework Ch.17 for Intent & Assurance management; Ch.4 for the determinism spectrum and the locked definitions of intent, eval, and assurance.

Break 4 — A4 fails: change is continuous, so governance must be too

What breaks. A4 held that the landscape is stable between governed cycles. Agents falsify this directly. They act, accumulate memory, encounter inputs no one anticipated, and drift — their behaviour shifts as their context, their model version, or their knowledge corpus changes, with no deployment event to mark the change. A quarterly architecture board governs a snapshot. Between two meetings, an agent population can take millions of actions, develop new failure modes, and drift away from the behaviour that was reviewed. **Gate-only governance is structurally blind to runtime behaviour** — it inspects the design and never sees the system act.

Why it breaks. A4 held because deployed systems did not change themselves. What was reviewed at a gate was what ran in production, unchanged until the next gate. Agentic systems change between gates — through learning, drift, model updates, and the sheer volume of unsupervised action — so a governance model that only meets at gates governs a system it can no longer see.

AEAF's answer. Dual governance. Design-time gates remain — they are still the right place to decide whether an allocation is sound, an autonomy level is justified, and an assurance case holds. But they are joined by **runtime governance**: continuous policy enforcement (guardrails), drift detection, incident response, and clear accountability for autonomous action. Governance becomes a **loop that runs while the system runs** — design-time decisions feed runtime enforcement, runtime evidence feeds the next design-time decision. → see AEAF Framework Ch.20 for continuous and runtime governance.

The four breaks, at a glance

ASSUMPTION	WHAT BREAKS IN THE AGENTIC ENTERPRISE	AEAF'S ANSWER	DEVELOPED IN
A1 Actors are human	Agents decide and act; the workforce is part human, part agent, part hybrid team. Charts and capability maps that assume human performers no longer describe the enterprise.	Reworked actor model (human / agent / hybrid); the blended workforce as the unit of design; capability allocation as a first-class act.	Ch.7, Ch.5
A2 Architecture is read by people	Agents read and obey architecture at runtime. A principle becomes a live constraint on an acting agent, not advice to a designer. Principles have gained a machine audience.	Dual-audience artifacts ; the Enforcement attribute (advisory / design-time / runtime); principles compiled into guardrails .	Ch.8, Ch.5
A3 Systems are deterministic & specifiable	A probabilistic actor cannot be specified or proven correct. "Correct" gives way to "acceptable within bounds, with known failure modes." Specify-and-test collapses.	Intent & Assurance replaces requirements-and-testing: declare intent, set guardrails, evaluate continuously.	Ch.17, Ch.4
A4 Change is episodic & gated	Agents act, learn, and drift at machine speed. A gate-only board is blind to runtime behaviour.	Dual governance : design-time gates plus runtime enforcement, drift detection, and continuous assurance.	Ch.20

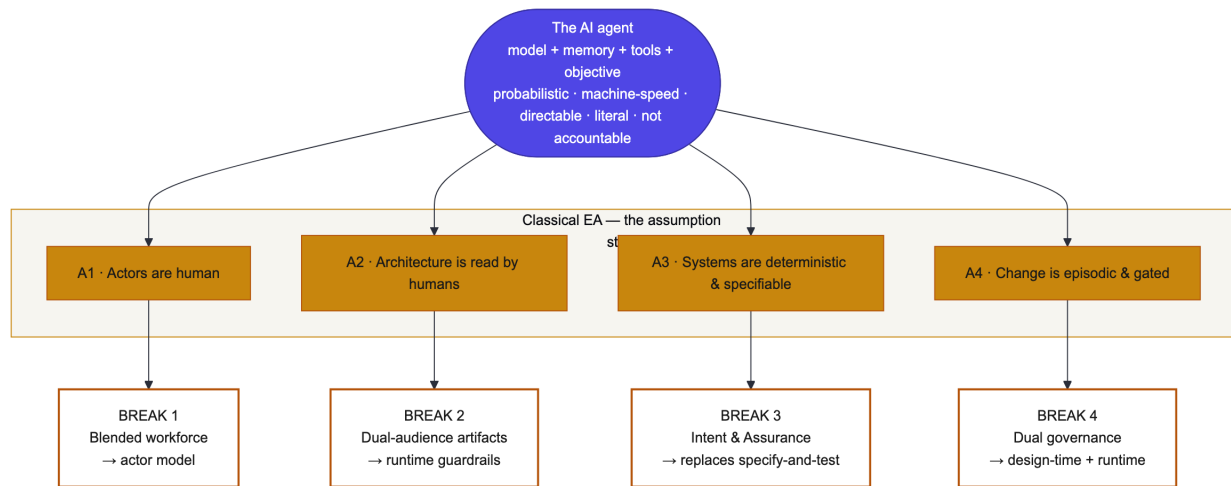


Figure 3.1 — The assumption-break map. The single new actor (indigo) strikes all four human-era assumptions (amber) at once; each break has a named AEAf response. The chapter's argument is that no one assumption can be patched in isolation, because the same actor falsifies all four — which is why AEAf reworks the framework rather than amending the method.

3.4 The new concern: trust, safety, and accountability as first-class architecture

The four breaks fix what the agentic enterprise damages. There is also something it *adds* — a concern with no classical analogue, which arises precisely because a non-human actor can now take consequential action.

When a non-human actor can act with consequence, the architecture must answer four questions it never had to ask before: who is accountable, what is the agent permitted to do, how is it overseen, and how do we detect and contain harm?

In classical EA these questions barely existed as *architecture*. A human performer carried their own accountability — the person who took the action was the person answerable for it. Permissions were access control, a technology concern. Oversight was line management, a people concern. Harm was an operational-risk matter handled after the fact. None needed a place in the architecture, because the human in the loop was the answer to all four at once.

An agent dissolves that convenience. It performs the work but bears none of the accountability; it can be permitted to do far more than any single human, far faster; it is overseen only to the degree the architecture designs oversight in; and it can cause harm at machine speed before a human notices. So trust, safety, and accountability stop being adjacent disciplines and become a **cross-cutting architecture concern, designed in rather than bolted on**:

- **Accountability** — every agent and every high-consequence action traces to exactly one named **accountability owner**, a human or legal entity. Accountability never transfers to the agent. An agent is *responsible* for execution; it is never *answerable* for outcomes.
- **Permission** — what an agent may do is an explicit, bounded grant (which tools, which actions, which limits), not an emergent consequence of what it can reach.
- **Oversight** — how a human relates to the agent's action is a designed control, in one of three defined modes (§3.5), chosen by consequence and confidence.
- **Containment** — guardrails, evals, drift detection, and incident response form a defence-in-depth so that when the agent is wrong — and a probabilistic actor will sometimes be wrong — the harm is bounded and detected.

This is why AEAF treats trust, safety, and accountability as a first-class, cross-cutting concern rather than a chapter at the back. → see AEAF Framework Ch.18 for the trust, safety, and accountability architecture; Ch.19 for the human-agent operating model.

3.5 The fused world: how work actually gets done

"Humans and agents working together" is easy to say and hard to design. The agentic enterprise is not *automation* — replacing one human task with one machine task and walking away. It is not *a chatbot bolted onto the side* of an unchanged organization. It is a **fabric of mixed teams** in which a given capability may be performed by a human, by an agent, or by both in concert — and where that allocation **moves over time** as agents grow more capable and as trust is earned or lost. Three ideas describe this operating picture, and AEAF makes each an explicit object of architecture.

1. Capability allocation. For every capability, the architecture must answer *who or what performs it* — human, agent, or hybrid team — and *under what oversight and at what autonomy level*. This is a first-class design decision recorded in the Capability Automation-Frontier Map and the Human-Agent RACI, not an implementation detail left to whoever builds the system.

2. The automation frontier. The boundary between human-performed and agent-performed work is not fixed. It advances as agents become more capable and as trust accrues; it occasionally retreats after an incident or as scope changes. AEAF treats this frontier as a thing to be **mapped, planned, and governed** — because moving it deliberately, rather than letting it drift, is the central transformation lever of the era.

3. Oversight modes. Human involvement is not binary. AEAF uses three named modes, and the right one depends on the consequence of the action, its reversibility, and the confidence the assurance evidence supports.

OVERSIGHT MODE	WHAT IT MEANS	WHEN IT FITS
In-the-loop	A human approves each action before it executes.	High consequence, low reversibility, or low confidence — the agent proposes, the human disposes.
On-the-loop	A human monitors a stream of agent actions and can intervene or halt.	Moderate consequence with good assurance — the agent acts, a human watches and can stop it.
Out-of-the-loop	No human in the runtime path; control is via guardrails and after-the-fact review.	Low consequence or high reversibility with strong guardrails and evals — the agent acts within bounds; humans review evidence, not actions.

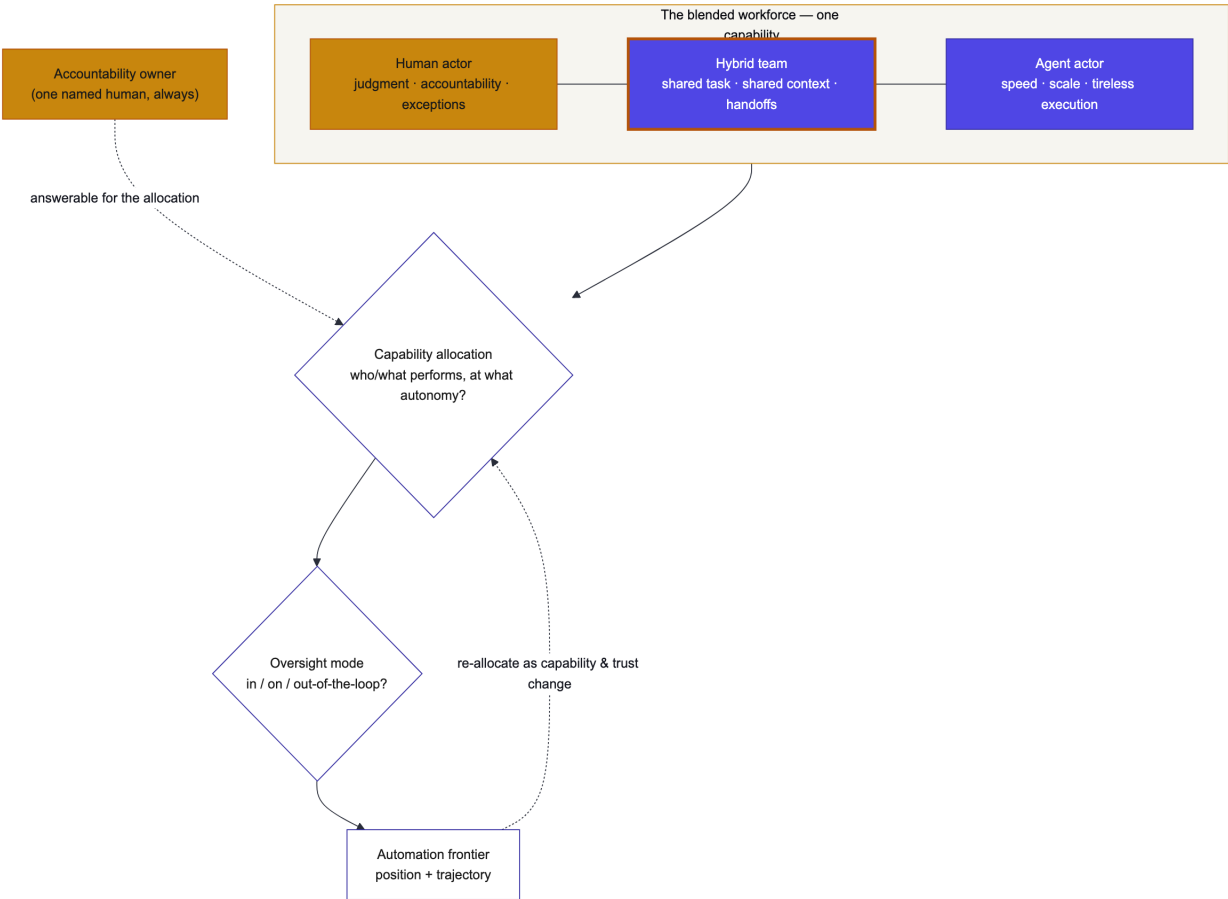


Figure 3.2 — The operating picture of the agentic enterprise. A capability is allocated across human (amber), hybrid (both), and agent (indigo) performers; the oversight mode is set by consequence and confidence; the frontier between human and agent work is planned and re-planned rather than left to drift; and one named human accountability owner remains answerable throughout. This is the world AEAF is designed to architect.

A worked illustration: the “resolve a refund” capability

Consider one stable, outcome-named capability — "resolve a customer refund request" — and watch it move across the frontier as trust is earned. The capability does not change. Who performs it, and how they are overseen, does.

STAGE	PERFORMER	AUTONOMY & BOUNDS	OVERSIGHT MODE	ACCOUNTABILITY OWNER
Today (baseline)	Human agent in the contact centre	Full human judgment within refund policy	n/a (human performs directly)	Head of Customer Operations

STAGE	PERFORMER	AUTONOMY & BOUNDS	OVERSIGHT MODE	ACCOUNTABILITY OWNER
First move	Hybrid team: agent drafts the resolution, human approves	Agent proposes refund + amount + reason; cannot execute	In-the-loop — human approves every refund before it issues	Head of Customer Operations
Frontier advances	Agent actor for the routine band, human for the rest	Agent issues refunds $\leq$ EUR 200 within policy autonomously; escalates anything above, any fraud signal, any policy edge case	On-the-loop — human monitors the stream, can halt; in-the-loop for escalations	Head of Customer Operations
Mature	Agent actor for the bulk; human for exceptions only	Agent issues refunds $\leq$ EUR 500 with strong guardrails (per-customer rate limit, duplicate-claim check, data-disclosure veto) and a passing eval suite	Out-of-the-loop within bounds; humans review evidence and handle escalations	Head of Customer Operations

Read down the table and the chapter's whole argument appears in one example. **A1:** the performer shifts from human to hybrid team to agent without the capability being redefined — the capability is actor-agnostic, which is exactly what lets the work move. **A2:** "never disclose another customer's data" is no longer advice to a designer; it is a runtime guardrail that vetoes the agent's action at the moment it would breach. **A3:** there is no test proving the agent always refunds correctly — there is an eval suite, a EUR 500 bound, and continuous assurance. **A4:** the EUR 200 → EUR 500 progression is not a one-time sign-off but a governed move on the frontier, justified by accumulated assurance evidence and reversible if drift or an incident appears. And throughout, the **accountability owner never changes** — the Head of Customer Operations is answerable at every stage, including when

the agent acts alone. The agent does the work; it never inherits the answerability. → see AEAF Framework Ch.7 for capability allocation and the frontier; Ch.11 for frontier mapping in the method.

3.6 What this demands of enterprise architecture

If agents are actors, the architecture acquires obligations it never had. Six follow directly from the four breaks and the new concern.

#	OBLIGATION	FOLLOWS FROM	REALIZED IN
1	Model the non-human workforce. Agents, the models behind them, the knowledge they draw on, and the guardrails that bound them are first-class entities in the metamodel — alongside, not buried inside, applications.	Break 1, A1	Ch.5 (metamodel)
2	Add a domain. Business, Data, Application, and Technology have no natural home for “the population of models and agents, their orchestration, their knowledge, their evaluation, and their autonomy.” AEAF adds a fifth domain — Intelligence & Agency , the +I in BDAT+I .	Break 1	Ch.6 (BDAT+I), Ch.15 (+I architecture)

#	OBLIGATION	FOLLOWS FROM	REALIZED IN
3	Make principles executable. Architecture an agent must obey at runtime has to express at least some principles as enforceable policy — guardrails — not only as prose.	Break 2, A2	Ch.8 (principles)
4	Replace verification with assurance. Since probabilistic behaviour cannot be proven correct, the architecture defines how behaviour is bounded, evaluated, monitored, and corrected as an ongoing discipline.	Break 3, A3	Ch.17 (Intent & Assurance)
5	Govern continuously. Governance reaches into runtime — policy enforcement, drift detection, incident response, and clear accountability for autonomous action — not only into design-time gates.	Break 4, A4	Ch.20 (continuous governance)

#	OBLIGATION	FOLLOWS FROM	REALIZED IN
6	Design the human-agent relationship. Allocation, oversight, escalation, and decision authority between people and agents are architecture, because they determine how the enterprise actually behaves.	New concern	Ch.7, Ch.19 (operating model)

None of these is optional in an enterprise that runs agents. An organization can run agents without doing any of them — most do today — but it then runs an agentic enterprise it has not architected, which is the condition AEAf exists to remedy.

3.7 AEAf’s design tenets

AEAf is meant to be used the way mature EA frameworks are used: **selected, tailored, and partially adopted**, not swallowed whole. The framework therefore holds itself to six tenets, each a direct response to something in this chapter.

TENET	WHAT IT COMMITS AEAf TO	ANSWERS
1. Answer TOGAF, don’t abandon it.	Keep the proven spine — governed baseline→target under iteration. Extend where assumptions bend; replace only where they break. Provide a migration path for existing TOGAF practices rather than asking architects to start over.	§3.1; the breaks are surgical, not wholesale
2. Two audiences, always.	Every normative artifact is written for a human and expressible for a machine. Dual-audience is the default, not an afterthought.	Break 2

TENET	WHAT IT COMMITS AEAFF TO	ANSWERS
3. Intent over specification.	Lead with declared intent, bounds, and evaluation. Treat exhaustive specification as neither possible nor required for probabilistic actors.	Break 3
4. Accountability is never delegated to an agent.	An agent can act; a named human or legal entity remains accountable. The architecture must always be able to answer "who is answerable for this?"	The new concern
5. Tailorable by design.	Modular, with an explicit map of what depends on what, so an organization can adopt a coherent subset and grow into the rest.	Real-world adoption
6. Evidence over hype.	Falsifiable claims, real trade-offs, named failure modes. The framework earns trust the way it asks agents to earn it — through evidence, not assertion.	The framework's own credibility

The first tenet deserves a final word, because it sets AEAFF apart from both the "throw out EA" and the "AI changes nothing" positions. Classical EA is not obsolete; its discipline of governed change from a known baseline to a designed target is exactly what an enterprise scattering agents across its operations most needs. What is obsolete is the unstated assumption stack beneath it. AEAFF's wager is that you keep the discipline and rebuild the premises — and that the result is a framework an architect who knows TOGAF can adopt without abandoning what already works.

The agentic enterprise already exists in fragments — a copilot here, an autonomous workflow there, an agent quietly issuing refunds under a limit no architecture ever set. What it lacks is an architecture. The rest of Book 1 builds one: the core concepts (Ch.4), the metamodel (Ch.5), the five domains (Ch.6), the actor model and blended workforce (Ch.7), dual-audience principles (Ch.8), and the method that ties them together (Ch.9 onward). This chapter has stated

the thesis those chapters realize — that the agent is a new kind of actor, that it breaks four assumptions classical EA never wrote down, and that the work of architecture is now to design, allocate, govern, and assure a workforce that is part human and part machine.

Part II • Foundations

Chapter 4 — Core concepts & definitions

In brief

This chapter is the framework's vocabulary. Everything after it — the meta-model, the domains, the principles, the AEAF Method — assumes these terms mean exactly one thing, so this chapter fixes them. The single most consequential definition is that an **actor** is no longer synonymous with a human: it specialises into a human actor, an agent actor, and a hybrid team. Around that sit the concepts the agentic enterprise cannot be architected without — the separation of **agent, model, and application**; the graded **autonomy spectrum** (L0 → L4); the **determinism spectrum** that replaces "correct" with "acceptable within bounds"; and the cluster of governance terms (**guardrail, policy, principle, intent, evaluation, assurance, oversight modes, escalation, accountability owner**) that make a non-human actor governable. Each definition here is canonical and reused verbatim throughout both books. This chapter **defines**; later chapters **apply**.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where this chapter contrasts the two, it does so to locate AEAF's additions precisely, not to characterise TOGAF.

How to read this chapter: the lexicon in §1 is the master reference table — scan it first, return to it often. The sections that follow expand the concepts that need more than a row: the actor model and the agent/model/application distinction (§2), the autonomy spectrum (§3), the determinism spectrum (§4), capability and the automation frontier (§5), the governance vocabulary (§6), and the two structural terms — the Intelligence & Agency domain and BDAT+I — that name where this all lives (§7).

1. The locked lexicon (master reference)

Every term below has a fixed meaning across both AEAF books. Use it exactly; do not coin synonyms. Terms marked (**home: Ch.X**) are defined in depth in the chapter named, and this chapter gives the working definition you need to read on. The "Why it matters to architecture" column is the point of the table: a glossary lists meanings, but architecture needs to know *what each term forces a design to account for*.

TERM	DEFINITION	WHY IT MATTERS TO ARCHITECTURE
Actor	Any entity that performs work in the enterprise. Specialises into human actor , agent actor , and hybrid team .	"Who performs this?" can no longer be answered "a person." Every capability allocation must say which kind of actor.
Human actor	A person, or an organisational role filled by people, who performs work and can hold accountability.	The only actor type that can be an accountability owner. Carries judgment, exception-handling, and answerability.
Agent actor	A software system that pursues a goal by reasoning over a model, taking actions through tools, drawing on a knowledge corpus, and operating within guardrails at a defined autonomy level.	The new first-class actor. Forces the metamodel to model models, tools, guardrails, evals, and autonomy as real objects.
Hybrid team	A unit of human and agent actors jointly performing one capability with shared context and defined handoff rules.	The realistic unit of work in the near term. Forces design of handoffs, shared context, and split accountability.
Agent	The actor: model + tools + instructions + memory + objective, running in a loop within guardrails.	The thing you govern. Distinct from the model it runs on (see §2).

TERM	DEFINITION	WHY IT MATTERS TO ARCHITECTURE
Model	The underlying machine-learning model (typically a large language or multimodal model) that supplies an agent's reasoning and generation. One model can power many agents.	Lifecycle, cost, and evaluation attach to the model independently of any one agent. Swapping a model is an architecture event.
Capability	A stable, outcome-named ability of the enterprise ("Settle a claim"), independent of who or what performs it.	Actor-agnostic by design — this is what lets work move between humans and agents without rewriting the capability model.
Capability allocation	The decision and record of which actors perform a given capability, at what autonomy level, under what oversight and guardrails.	The central AEAF design act. Allocation, not org-charting, is where human/agent work is decided. (home: Ch.7)
Automation frontier	The current boundary, per capability, between human-performed and agent-performed work, and the direction that boundary is expected to move.	A frontier moves. Treating it as a planned, governed trajectory is the central transformation lever. (home: Ch.7)
Autonomy level	The graded degree of independent action granted to an agent actor for a capability, from suggest-only (L0) to autonomous-within-bounds (L4).	Sets the required controls and the oversight mode. An ungraded "is it automated?" question is no longer sufficient. (home: §3; applied Ch.15)
Determinism spectrum	The range from fully deterministic behaviour (same input → same output, specifiable and testable) to non-deterministic behaviour (probabilistic, bounded and evaluated rather than specified).	Tells you which assurance model applies to a component: testing, evaluation, or both. (home: §4)

TERM	DEFINITION	WHY IT MATTERS TO ARCHITECTURE
Guardrail	A constraint that bounds an agent's behaviour at runtime — input filter, output check, tool permission, spend limit, action veto — enforced regardless of the agent's reasoning.	The mechanism that makes a probabilistic actor safe to deploy. Bounds; does not specify. (home: §6)
Policy (machine-enforceable)	A rule expressed so it can be enforced automatically at design time or runtime, not only read and obeyed by a human.	The bridge from human-readable governance to runtime enforcement. A policy is the rule; a guardrail is one mechanism that enforces it.
Principle	A foundational architecture statement carrying an Enforcement attribute: advisory, design-time, or runtime.	Principles now have a second audience — agents at runtime. The Enforcement attribute makes that explicit. (home: Ch.8)
Intent	The bounded statement of what outcome an actor is to achieve and within what limits — replacing the fully-specified requirement for non-deterministic work.	You cannot fully specify a probabilistic actor; you declare intent and bounds, then evaluate. Pairs with assurance. (home: Ch.17)
Evaluation (eval) / eval suite	Evaluation measures an agent's behaviour against expected outcomes on a defined set of cases. An eval suite is the maintained collection of such cases for a capability.	Replaces one-time acceptance testing for behaviour that cannot be exhaustively specified. (home: §6; applied Ch.17)
Assurance / assurance case	Assurance is the continuing evidence that an agent behaves within its intent and bounds. An assurance case is the structured argument and evidence that it is fit to operate.	Assurance is ongoing, not a sign-off event. It is how you defend an autonomous capability to an auditor or a board. (home: Ch.17)

TERM	DEFINITION	WHY IT MATTERS TO ARCHITECTURE
Oversight modes	How a human relates to an agent's actions: in-the-loop (approves each action), on-the-loop (monitors, can intervene), out-of-the-loop (no human in the runtime path; control via guardrails and after-the-fact review).	The architectural lever that balances speed against consequence. Tied to the autonomy level. (home: §6; applied Ch.19)
Escalation	The defined path by which an agent hands control to a human (or higher-authority actor) on hitting a bound, low confidence, or a guardrail veto.	A designed path, not a failure. Specifies trigger, target, and fallback. (home: §6; applied Ch.19)
Accountability owner	The single named human or role answerable for the outcomes of a capability, including work done by agent actors. Accountability never transfers to an agent.	Exactly one per capability. The architecture must always answer "who is answerable for this?" (home: §6)
Knowledge corpus	The governed body of content (documents, data, embeddings) an agent retrieves from to ground its reasoning, with defined sources, freshness, and access controls.	The cognitive substrate. Its freshness and access controls are now an architecture concern, not an implementation detail.
Intelligence & Agency domain	The fifth AEAF architecture domain, covering models, agents, knowledge corpora, guardrails, evals, autonomy levels, and oversight.	The home for everything that has no natural place in Business, Data, Application, or Technology. (home: §7; applied Ch.6, Ch.15)
The AEAF Method	The phased, cyclic method for developing and governing an Agentic Enterprise architecture.	AEAF's answer to the TOGAF ADM. Defines how the concepts above are produced and kept current. (home: Ch.9)

TERM	DEFINITION	WHY IT MATTERS TO ARCHITECTURE
BDAT+I	The five AEAF domains: B usiness, D ata, A pplication, T echnology, plus Intelligence & Agency.	The “+I” is AEAF’s defining structural addition to the classical four-domain EA stack. (home: §7; applied Ch.6)

Table 4.1 — The AEAF locked lexicon. The remainder of this chapter expands the rows that need more than a sentence.

2. Actor, and the agent / model / application distinction

2.1 Actor is no longer a synonym for human

Classical enterprise architecture treats *actor* and *human* as interchangeable: an actor performs a function, holds a role, follows a process, reads the architecture. Every one of those statements assumed a person. The agentic enterprise breaks the assumption, so AEAF makes **actor** an abstract umbrella with three concrete subtypes.

SUBTYPE	WHAT IT IS	DISTINGUISHING PROPERTIES	EXAMPLE
Human actor	A person or a role filled by people.	Judgment, initiative, intrinsic accountability, slow, expensive, scarce.	A claims handler who decides a disputed €40,000 settlement.
Agent actor	A model-driven software actor with tools, memory, an objective, and an autonomy level.	Probabilistic, fast, tireless, literal, no intrinsic accountability.	An agent that auto-settles undisputed claims under €1,000 within policy.

SUBTYPE	WHAT IT IS	DISTINGUISHING PROPERTIES	EXAMPLE
Hybrid team	Humans and agents performing one capability together.	Shared context, defined handoffs, split responsibility, single accountability owner.	An agent triages and drafts every claim; a human approves anything it flags as ambiguous.

Table 4.2 — The three actor subtypes. The agent actor and hybrid team are new to EA; the human actor is the classical actor, now named explicitly so it can be contrasted.

Why this matters to architecture: once actor specialises, the question "who performs this capability?" can no longer be answered before asking "*which kind* of actor, at what autonomy level, under what oversight?" That chain — capability → allocation → actor + autonomy + oversight → accountability owner — is the backbone the rest of the framework hangs on (→ see AEAF Framework §5).

2.2 Three things people call "the AI": agent, model, application

The most common modelling error in early agentic systems is to draw one box and label it "the AI." There are three separate things, with different lifecycles, owners, and governance — and conflating them makes the architecture un-governable.

CONCEPT	WHAT IT IS	OWNS / CARRIES	LIFECYCLE EVENT
Model	The reasoning engine — a trained large language or multimodal model.	Provider, version, context window, cost-per-token, risk profile.	A new model version is released; cost or capability changes.
Agent	An actor built on a model: model + instructions + tools + memory + objective + guardrails + autonomy level.	Purpose, permitted actions, guardrails, accountability owner, eval suite.	The agent is granted a new tool, or its autonomy level is raised.
Application	The runtime that hosts the agent — the deployable service, its APIs, scaling, and integration points.	Deployment, availability, observability, integration contracts.	The service is redeployed, scaled, or re-platformed.

Table 4.3 — Model, agent, and application are three entities, not one. One model can power many agents; one application can host many agents; one agent uses one model at a time but can be re-pointed to another.

A worked example makes the separation concrete. A travel-booking agent named *TripBooker* runs on the model *GPT-class-v4*, is hosted by the *Concierge* application, and is permitted to book travel within a €2,000 per-trip limit at autonomy level L3.

- Swap *GPT-class-v4* for a cheaper model: the **model** changed; *TripBooker* and *Concierge* are unchanged in identity, but its eval suite must re-run.
- Give *TripBooker* a new "rebook cancelled flights" tool: the **agent** changed; the model and the hosting application did not.
- Move *Concierge* to a new region for data residency: the **application** changed; the agent and model did not.

Each of these is a different change, reviewed by a different owner, against a different artifact (the Model Portfolio, the Agent Catalog, the application's deployment record). Collapsing them into "we changed the AI" loses all three traceability paths.

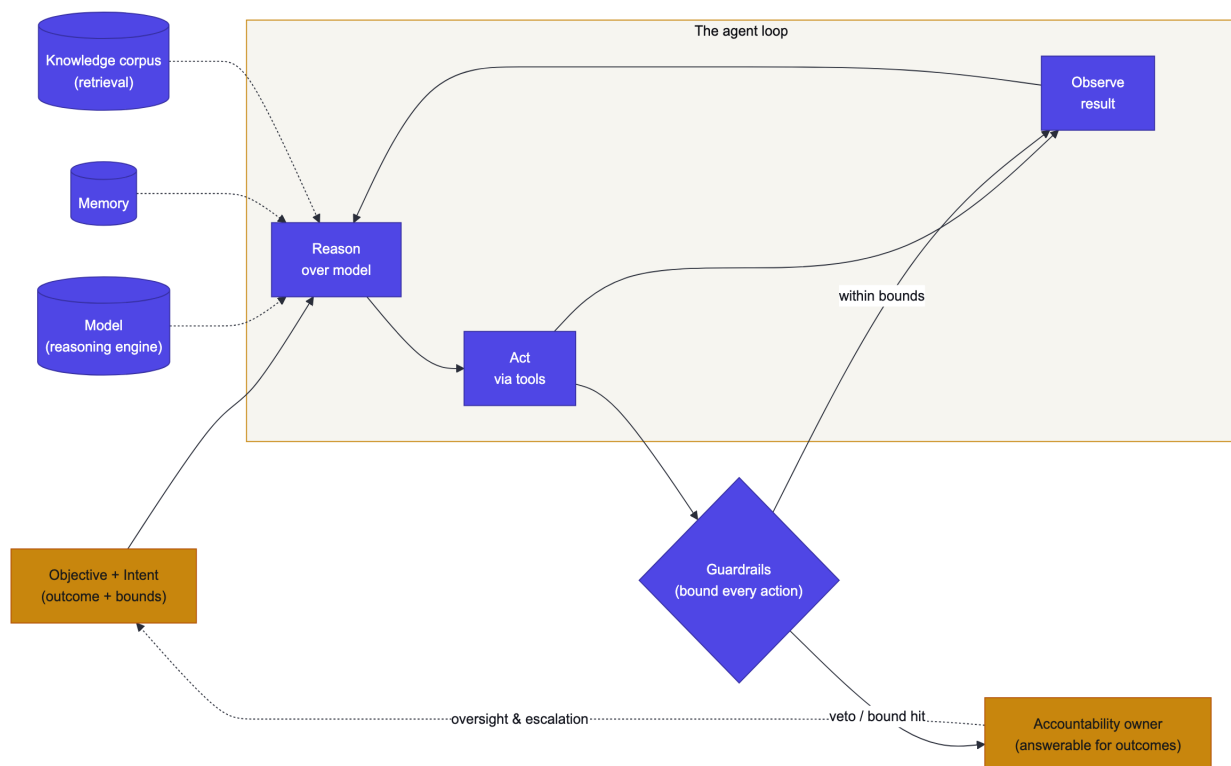


Figure 4.1 — An agent is a model placed in a loop with memory, tools, a knowledge corpus, and an objective, bounded by guardrails. Indigo nodes are machine intelligence; amber nodes are human judgment — the objective and bounds the human sets, and the

accountability owner who receives every guardrail veto. The model (indigo) is one component inside the agent, not the agent itself. This is the picture §2.2 separates into three governed entities.

3. The autonomy spectrum

3.1 Why autonomy is graded, not binary

"Is the work automated?" is a yes/no question that the agentic enterprise cannot answer usefully. The same agent can *suggest* a draft (a human does everything), *act and report* (a human reviews after the fact), or *act freely within a spend limit* (no human in the path). These are different risk profiles, different control requirements, and different oversight modes — for the same agent on the same capability. AEAF therefore grades autonomy on a five-level scale, the **Autonomy-Level Classification**.

The scale is the canonical reference; do not invent ad-hoc levels. Each level pairs an agent behaviour with a human role, a minimum set of required controls, and a default oversight mode. Higher levels are not "better" — the right level for a capability is set by consequence, reversibility, and assurance evidence, which Chapter 15 applies in detail.

LEVEL	NAME	WHAT THE AGENT DOES	HUMAN ROLE	REQUIRED CONTROLS (MINIMUM)	DEFAULT OVERSIGHT MODE	EXAMPLE
L0	Assistive	Suggests, drafts, retrieves; takes no action of its own.	Decides and acts on everything .	Output disclosure ("AI-generated"); source attribution .	In-the-loop (by construction).	Drafts a reply a human edits and sends.
L1	Recommended	Proposes a specific action and the reasoning for it; does not execute.	Approves or rejects each proposed action.	Confidence surfaced; clear approve/reject; full audit log.	In-the-loop.	Recommends a refund amount; a human clicks approve.

LEVEL	NAME	WHAT THE AGENT DOES	HUMAN ROLE	REQUIRED CONTROLS (MINIMUM)	DEFAULT OVERSIGHT MODE	EXAMPLE
L2	Execute-with-approval	Prepares and stages an action; executes only after explicit human sign-off.	Approves each action before it executes.	Pre-execution preview; reversible staging; per-action veto.	In-the-loop.	Stages a supplier payment; a human authorises release.
L3	Act-and-report	Executes within bounds without prior approval; reports every action for monitoring.	Monitors a stream; can intervene or roll back after the fact.	Runtime guardrails; spend/rate limits; full action log; reversibility or compensation; escalation on low confidence.	On-the-loop.	Auto-settles undisputed claims under €1,000; flags the rest.
L4	Autonomous-within-bounds	Acts freely toward its objective inside hard guardrails; no human in the runtime path.	Sets bounds, reviews assurance evidence periodically; not in the action path.	Hard runtime guardrails with veto; continuous evals; drift detection; incident response; defined escalation; assurance case.	Out-of-the-loop.	Re-prices low-margin SKUs hourly within a ±5% band.

Table 4.4 — The AEAF Autonomy-Level Classification. "Required controls" are cumulative: every level inherits the controls of the levels below it. Note that "autonomous" (L4) means autonomous *within bounds*, never unbounded — there is no level at which guardrails are removed.

Two properties of the scale carry weight downstream:

- **Controls are cumulative.** Each level adds to the controls of the level beneath it. You cannot operate at L3 without the L0–L2 controls already in place.
- **The level is per capability, not per agent.** The same agent may run at L1 for high-value refunds and L3 for small ones. Autonomy is a property of the *allocation*, recorded against the capability (→ see AEAF Framework §7).

3.2 How an agent moves up the scale

An agent does not start at L4. It earns autonomy as assurance evidence accumulates and is demoted when that evidence degrades. The progression is a governed lifecycle, not a configuration flag.

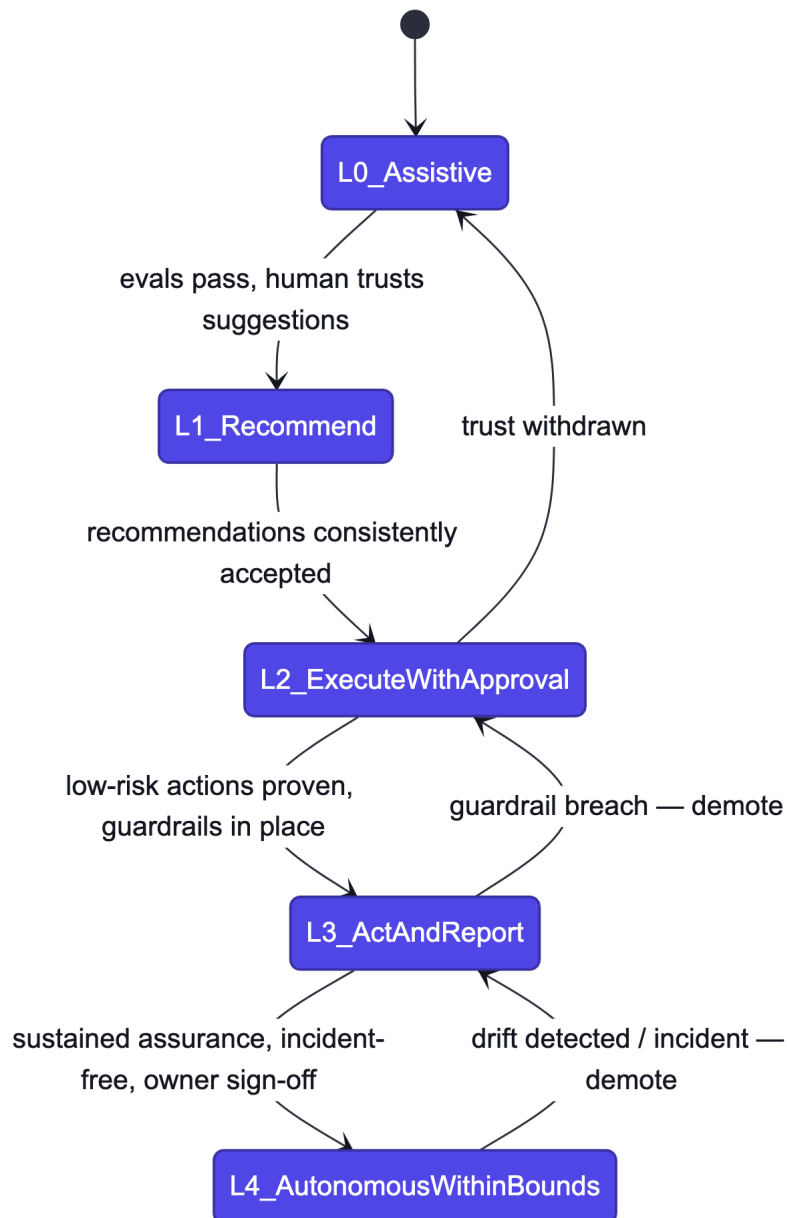


Figure 4.2 — The autonomy progression is bidirectional. An agent is promoted upward only on accumulated assurance evidence and an accountability owner's sign-off (amber gates in prose; the transitions); it is demoted automatically on drift or a guardrail breach. Autonomy is granted, not assumed, and it is revocable. Chapter 15 specifies the promotion and demotion gates.

4. The determinism spectrum

4.1 From "correct" to "acceptable within bounds"

Classical systems are deterministic: the same input produces the same output, the behaviour can be specified before it is built, and it can be verified by testing. A correct system conforms to its specification, and conformance is checkable case by case. Agent actors break this. A large language model is **probabilistic** — the same prompt can yield different actions — so its behaviour cannot be fully specified up front and cannot be exhaustively tested. The assurance question changes from *"is the output correct?"* to *"is the behaviour acceptable, within stated bounds, often enough, with known failure modes?"*

This is not a binary of "deterministic systems vs AI." Most real components sit somewhere on a spectrum, and an agent system usually spans it: a deterministic guardrail wraps a probabilistic core, which retrieves from a deterministic data store. AEAF asks an architect to state *where each component sits* and to apply the matching assurance model.

PROPERTY	DETERMINISTIC END	PROBABILISTIC END
Input → output	Same input, same output.	Same input, possibly different output.
Specifiability	Behaviour specified up front.	Behaviour declared as intent + bounds; not fully specifiable.
Standard of "good"	Correct (conforms to spec).	Acceptable (within bounds, above an eval threshold, with known failure modes).
Assurance method	Testing — verify defined cases pass.	Evaluation — measure behaviour across a case suite; monitor continuously.
When behaviour changes	Only when code changes (episodic).	Can change with model, prompt, data, or drift (continuous).
Example component	A tax-calculation rule engine.	An agent classifying a free-text complaint.

Table 4.5 — The determinism spectrum, end to end. The shift from "correct" to "acceptable within bounds" is the reason AEAF replaces requirements-and-testing with intent-and-assurance for the probabilistic end (→ see AEAF Framework §17).

4.2 Why this is an architecture decision, not a coding one

Where a component sits on the spectrum dictates how it is governed, so it must be decided at design time. A deterministic rule engine gets a test plan and a pass/fail gate. A probabilistic agent gets an intent statement, guardrails, an eval suite, and continuous assurance. Putting a probabilistic component behind a "tested once, signed off" gate is a category error — it will pass the gate and then drift in production. Conversely, demanding continuous evaluation of a deterministic rule engine wastes effort on behaviour that cannot vary.

A short example: an agent that decides loan eligibility (probabilistic) is fronted by a hard guardrail that rejects any applicant under the legal age (deterministic). The guardrail is *tested* — it either blocks under-age applicants or it is broken. The agent is *evaluated* — measured across thousands of cases for fairness, accuracy, and acceptable failure modes, then monitored for drift. Same feature, two assurance models, because two components sit at two ends of the spectrum.

5. Capability, capability allocation, and the automation frontier

These three terms form a chain, and the chain is where AEAF does its central design work. They are defined briefly here; Chapter 7 applies them.

Capability is a stable, outcome-named ability of the enterprise — "Settle a claim," "Onboard a supplier," "Reconcile an invoice." A capability is deliberately **actor-agnostic**: it names *what* the enterprise can do, never *who* does it. This is not a stylistic choice. It is precisely the actor-agnosticism that lets work be reallocated between humans and agents without rewriting the capability model. If "Settle a claim" were defined as "a claims handler settles a claim," moving it to an agent would force a model change; defined outcome-first, the allocation simply changes.

Capability allocation is the decision and record of *which actors* perform a given capability, at *what autonomy level*, under *what oversight and guardrails*. This is the central AEAF design act — not staffing, not resourcing. A single allocation answers four questions at once: who/what performs this, how autonomously, under whose oversight, and who is accountable. One capability may carry several allocations over time, or split by sub-case (small claims to an agent at L3, large claims to a hybrid team at L1).

Automation frontier is the boundary, per capability, between human-performed and agent-performed work — and the direction it is expected to move. The word "frontier" is chosen deliberately: it is a line that *moves*, advancing as agents prove capable and trust is earned, occasionally retreating after an incident. AEAF treats the frontier as something to be **mapped, planned, and governed** across the whole capability model, captured in the Capability Automation-Frontier Map. Describing the frontier as a fixed percentage ("40% automated") misses the point; what matters is its position per capability and its trajectory.

Worked example. *Invoice reconciliation* is a capability. Last year its allocation was: human accountant performs it, agent at L0 (assistive, suggests matches). This year the allocation moved: agent at L3 performs routine matches and reports; a human handles exceptions in a hybrid team. The **automation frontier** for *invoice reconciliation* advanced from "agent assists" to "agent acts, human handles exceptions" — a planned move, recorded with a date and the assurance evidence that justified it, not a drift that happened by accident.

6. The governance vocabulary

The terms in this section are what make a non-human actor governable. They are introduced here as definitions; their catalog (principles, Ch.8), their assurance machinery (intent and evals, Ch.17), and their operating model (oversight and escalation, Ch.19) are developed later.

6.1 Guardrail, policy, and principle — three layers, not synonyms

These three are routinely conflated and are not the same thing. They form a layered chain from human belief to runtime enforcement.

TERM	WHAT IT IS	AUDIENCE	ENFORCEMENT
Principle	A foundational architecture statement of what the enterprise believes and requires, carrying an Enforcement attribute.	Humans (always); agents (when runtime).	Advisory, design-time, or runtime — declared per principle.

TERM	WHAT IT IS	AUDIENCE	ENFORCEMENT
Policy (machine-enforceable)	A rule expressed so it can be enforced automatically, derived from a principle, standard, or regulation.	Machines (and humans who read it).	Enforced at design time or runtime.
Guardrail	A concrete mechanism that bounds an agent's behaviour at runtime, enforcing a policy regardless of the agent's reasoning.	Machines, at runtime.	Runtime, on every relevant action.

Table 4.6 — Principle → policy → guardrail. A principle is *what we believe*; a policy is *the rule expressed enforceably*; a guardrail is *the mechanism that enforces it on a live agent*. One principle may compile into several guardrails.

The chain in one example. The **principle** "Customer PII is never sent to third parties" carries Enforcement = runtime. It is expressed as a **policy** — "No outbound tool call may contain unmasked PII fields." That policy is enforced by a **guardrail** — an output filter that inspects every tool call the agent makes and vetoes any containing unmasked PII, regardless of what the agent decided to do. In classical EA the principle was advice to an engineer reading documentation; in AEAF it must reach the agent at runtime, which is why principles now carry an Enforcement attribute (→ see AEAF Framework §8).

A guardrail **bounds**; it does not **specify**. It does not tell the agent what to do; it constrains what the agent is allowed to do — an input filter, an output check, a tool permission, a spend limit, an action veto. This distinction matters because it is what lets a probabilistic actor be deployed safely: you cannot specify its every action, but you can bound them.

6.2 Intent versus requirement

A **requirement** specifies *what to build and how it must behave*, case by case, and is verified by testing. It is the right tool for deterministic work. An **intent** declares *what outcome to achieve and within what limits* — and deliberately does not enumerate the steps. It is the right tool for probabilistic work, where the steps cannot be fully specified in advance.

The contrast in one capability — "respond to a billing dispute":

- **As a requirement:** "On a dispute, retrieve the invoice, compare to the contract rate, if mismatch issue a credit equal to the difference, send template letter B." Every branch specified; testable.
- **As an intent:** "Resolve billing disputes fairly and within policy. Outcome: the customer's correct balance and a clear explanation. Bounds: never credit more than the disputed amount; never exceed a €500 goodwill limit without escalation; always cite the contract clause." The outcome and limits are stated; the path is left to the agent and measured by evals.

Intent pairs with assurance: intent says what good looks like and where the edges are; assurance is the evidence that the agent stays inside them (→ see AEAF Framework §17).

6.3 Evaluation, eval suite, assurance, and assurance case

For deterministic work you **test** — run defined cases, each passes or fails, sign off once. For probabilistic work you cannot prove correctness, so AEAF substitutes a measurement-and-evidence discipline.

TERM	DEFINITION	REPLACES / EXTENDS
Evaluation (eval)	The measurement of an agent's behaviour against expected outcomes on a defined set of cases.	The one-time pass/fail test, for behaviour that cannot be exhaustively specified.
Eval suite	The maintained collection of eval cases for a capability, with expected outcomes and pass thresholds.	The test plan — but living, re-run on every model or prompt change.
Assurance	The continuing evidence — from evals, guardrail logs, and oversight — that an agent behaves within its intent and bounds over time.	One-time acceptance testing; assurance is ongoing, not an event.
Assurance case	The structured argument and evidence that an agent is fit to operate within its bounds, with residual risk stated.	The sign-off memo — but a living argument, reviewed and dated.

Table 4.7 — The assurance vocabulary. The defining shift: a test is a moment; assurance is a continuing state. An agent that passed its evals in March can drift by June, which is why assurance is monitored, not signed off.

Example. A claims-triage agent has an **eval suite** of 2,000 historical claims with known correct dispositions; an **evaluation** run scores the agent at 97% agreement with bounded failure modes. Its **assurance** is the standing dashboard of eval scores, guardrail vetoes, and escalations that shows it staying within bounds week to week. Its **assurance case** is the document an auditor reads: "this agent operates at L3 because evals hold above 95%, no guardrail breaches in 90 days, all sub-€1,000 claims, full reversibility, residual risk X accepted by owner Y."

6.4 Oversight modes, escalation, and the accountability owner

These three define the human-agent relationship at runtime.

Oversight modes name how a human relates to an agent's actions. There are exactly three; use them as named, not "human-in-the-loop" as a catch-all.

MODE	HUMAN RELATIONSHIP TO THE ACTION	PAIRS WITH AUTONOMY	WHEN TO USE
In-the-loop	Approves each action before it executes.	L0–L2	High consequence, low reversibility, or thin assurance.
On-the-loop	Monitors a stream of actions; can intervene or roll back.	L3	Moderate consequence, actions reversible or compensable.
Out-of-the-loop	Not in the runtime path; control is by guardrails and after-the-fact review.	L4	Low consequence per action, hard guardrails, strong assurance.

Table 4.8 — The three oversight modes and the autonomy levels they pair with. The mode is an architectural choice set by consequence, reversibility, and confidence (→ see AEAF Framework §19).

Escalation is the defined path by which an agent hands control to a human (or a higher-authority actor) when it hits a bound, has low confidence, or is vetoed by a guardrail. An escalation is a *designed path, not a failure* — a well-architected agent escalates often and

cleanly. Every escalation specifies three things: the **trigger** (what causes it), the **target** (who receives it), and the **fallback** (what happens to the in-flight work meanwhile). An agent that hits its €500 goodwill limit does not fail; it escalates to a named human with the case context attached.

Accountability owner is the single named human or role answerable for the outcomes of a capability, including all work done by agent actors on it. Two rules are absolute and run through the whole framework:

- **Exactly one accountability owner per capability.** Diffuse accountability is no accountability.
- **Accountability never transfers to an agent.** An agent can be *responsible* for execution — it does the work — but a human or legal entity is always *accountable* for the outcome. The architecture must, at every point, be able to answer "who is answerable for this?" This is why the accountability owner anchors every agent in the metamodel (→ see AEA Framework §5) and why it is recorded in the Human-Agent RACI and the Trust & Accountability Matrix.

7. The two structural terms: the Intelligence & Agency domain and BDAT+I

The concepts above need a home in the architecture, and the classical four domains do not have one. **BDAT** — Business, Data, Application, Technology — is the standard decomposition of an enterprise architecture. None of its four domains has a natural place for *the population of models and agents, the knowledge they draw on, the guardrails that bound them, the evals that assure them, and the autonomy levels they operate at*. Forcing agents into the Application domain is the modelling error AEA exists to correct: an agent is an actor, not an application.

AEA therefore adds a fifth domain — **Intelligence & Agency**, abbreviated **+I**, giving the five domains **BDAT+I** (Business, Data, Application, Technology, plus Intelligence & Agency). The +I domain owns models, agent actors, knowledge corpora, guardrails and policies, eval suites, autonomy levels, and oversight controls. It is a *domain*, sitting alongside the other four — not a layer inside the Application domain and not "the AI domain." It is where the agent/model distinction (§2), the autonomy and determinism spectra (§3–4), and the governance vocabulary (§6) are catalogued and governed.

The +I domain does not stand alone; it joins to the classical four through shared entities that already appear in this chapter:

+I ELEMENT	JOINS TO	THROUGH
Agent actor	Business	Capability allocation — the agent performs a business capability.
Knowledge corpus	Data	A corpus is curated from data entities for machine cognition.
Agent runtime	Application / Technology	The application hosts the agent; the technology platform runs the model.
Guardrail / policy	Business / Technology	A policy realises a business principle; a guardrail is enforced at a technology point.

Table 4.9 — The +I domain's join points to BDAT. The Intelligence & Agency domain is additive, not a silo; Chapter 6 develops these relationships in full.

Finally, **the AEA Method **** is the phased, cyclic method that produces and keeps current everything defined in this chapter — the allocations, the catalogs, the principles, the assurance evidence. Where TOGAF uses the ADM, AEA uses the AEA Method. It is named here only to place it; Chapter 9 defines it.

8. What this chapter fixed, and where each concept is applied

This chapter is the definitional floor. The terms are now locked; the rest of Book 1 builds on them without redefining them.

CONCEPT DEFINED HERE	APPLIED IN
Actor model; agent/model/application distinction	The meta-model — → see AEA Framework §5

CONCEPT DEFINED HERE	APPLIED IN
Intelligence & Agency domain; BDAT+I	The five domains — → see AEA Framework §6
Capability, allocation, automation frontier	The reworked actor model & blended workforce — → see AEA Framework §7
Principle and its Enforcement attribute	Principles for the Agentic Enterprise — → see AEA Framework §8
The AEA Method	Method overview & the cycle — → see AEA Framework §9
Autonomy spectrum (L0–L4)	Intelligence & Agency architecture — → see AEA Framework §15
Intent, eval, assurance, assurance case	Intent & Assurance management — → see AEA Framework §17
Oversight modes, escalation, accountability owner	The human–agent operating model — → see AEA Framework §19

Table 4.10 — Where each concept defined in this chapter is applied. This chapter defines; the named chapters apply.

Chapter 5 — The AEAF meta-model

In brief

A content meta-model defines the *things* an enterprise architecture is allowed to describe and the *relationships* it can record between them. It is the schema behind every catalog, matrix, and diagram an architecture team produces. AEAF keeps the classical entities — Actor, Role, Function, Process, Service, Application Component, Data Entity, Technology Component, Driver/Goal/Objective, and Building Block — and adds the entities the agentic enterprise cannot be described without: **Agent Actor**, **Model**, **Knowledge Corpus**, **Guardrail**, **Intent**, **Eval Suite**, **Capability Allocation**, **Oversight Control**, **Autonomy Level**, **Automation Frontier**, **Accountability Owner**, **Action/Tool**, and **Assurance Case**. Three structural moves make the difference: **Actor stops meaning "human"**, **Model**, **Agent**, and **Application become three separate entities** rather than one, and a **Principle compiles into a Guardrail** when it must be enforced at runtime. The meta-model exists for one reason above all others — it makes the framework's promises *checkable*, because every claim AEAF makes about governance, accountability, and assurance reduces to a traceability chain the schema must be able to hold.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. This chapter contrasts AEAF against the TOGAF content metamodel to be precise about what changes; the comparison is descriptive, not a claim of lineage.

This chapter defines the entities and relationships. The artifacts that instantiate them — the Agent Catalog, the Trust & Accountability Matrix, and the rest — are specified in Chapter 21 and templated in Book 2. The principle catalog lives in Chapter 8; the method phases that populate the schema live in Chapter 9. → see AEAF Framework §21, §8, §9.

5.1 What a content meta-model is, and why it is the backbone

An architecture description is a set of statements: *this role performs that function, this application realizes that service, this data entity is owned by that business unit*. A **content meta-model** is the grammar of those statements. It declares the entity types you may instantiate (Role, Function, Service...), the attributes each entity carries, and the relationship types that may connect them. Everything a repository stores, everything a catalog lists, everything a matrix cross-tabulates, is an instance of the meta-model. Change the meta-model and you change what the architecture is *able to say*.

This is why the meta-model — not the method, not the principles — is the true backbone of a framework. The method tells you how to fill the repository; the principles tell you what good looks like; but the meta-model determines whether a question can even be *asked* of the architecture. If "which actor performs this capability?" can only be answered with a human name, the architecture is structurally blind to agents. If there is no entity for "the evidence that this agent is fit to operate," then assurance is a slide, not an architecture object, and it cannot be queried, versioned, or audited.

AEAF makes a strong claim in its founding thesis: that the agentic enterprise needs governance that reaches into runtime, accountability that never transfers to a machine, and assurance that replaces one-time verification (→ see AEAF Framework §3). Those are promises. A promise is only credible if it is *checkable*. The meta-model is what makes them checkable, because each promise reduces to a chain of entities and relationships that must be navigable end to end:

PROMISE FROM THE THESIS	THE CHAIN THAT MAKES IT CHECKABLE
"We can always answer who is accountable for an agent's action."	Action → Agent Actor → Accountability Owner must be traversable for every consequential Action.
"Principles agents must obey are enforced, not merely documented."	Principle (enforcement = runtime) → Guardrail → enforcement point must exist for every runtime principle.
"We govern continuously, not only at gates."	Model/Agent → Eval Suite → Assurance Case → Incident → corrective change must form a loop, not a line.

PROMISE FROM THE THESIS	THE CHAIN THAT MAKES IT CHECKABLE
"Work can move between humans and agents under control."	Capability → Capability Allocation → {Actor, Oversight Control, Autonomy Level} must be one navigable record.

If any link in these chains has no entity to hold it, the corresponding promise is unverifiable. The rest of this chapter defines the entities precisely so the chains in §5.5 close.

A note on what a meta-model is *not*. It is not the data (the specific agents, models, and capabilities of a given enterprise — those are instances). It is not the artifacts (the catalogs and matrices — those are *views* over instances, specified in Chapter 21). And it is not the method (the phases that create and maintain instances — Chapter 9). The meta-model is the type system beneath all three.

5.2 The core extension at a glance

The diagram below is the heart of the chapter: the entities AEAF adds and changes, and how they relate. Read it in three bands. The **machine-intelligence additions** (Agent Actor, Model, Knowledge Corpus, Guardrail, Eval Suite, Action) are the population the +I domain owns. The **human-agent join points** (Capability Allocation, Oversight Control, Autonomy Level, Automation Frontier) are where the blended workforce is designed. The **accountability anchor** (Accountability Owner) ties every agent back to a human or legal entity that answers for it. The indigo/amber convention applies throughout the chapter's diagrams: indigo for machine/agent entities, amber for human/judgment entities.

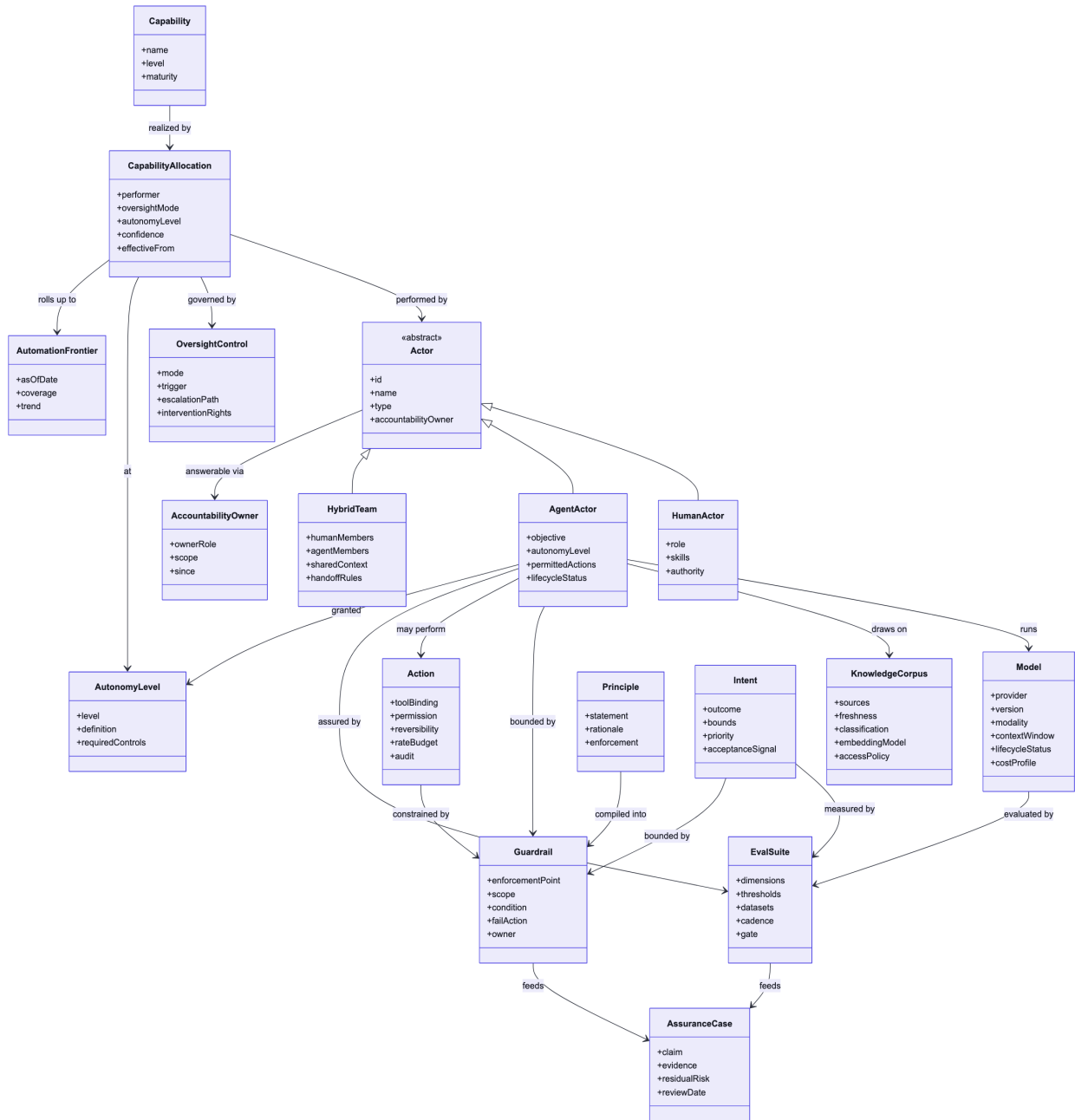


Figure 5.1 — The AEAF meta-model core. Three bands: the machine-intelligence additions an agent depends on (Model, Knowledge Corpus, Guardrail, Action, Eval Suite); the human-agent join points (Capability Allocation, Oversight Control, Autonomy Level, Automation Frontier) where the blended workforce is designed; and the accountability anchor that every Actor — human or agent — resolves to. The de-anthropomorphized Actor at the top is the single most consequential change; the Principle → Guardrail edge is the formal expression of architecture's second audience.

Figure 5.1 shows the additions in isolation for clarity. The retained TOGAF entities (Role, Function, Process, Service, Application Component, Data Entity, Technology Component, Driver/Goal/Objective, Building Block) connect into this picture at the points the catalog in §5.3 names — most importantly, an Application Component *hosts* an Agent Actor rather than *being* one, and a Data Entity may *feed* a Knowledge Corpus.

5.3 The full entity catalog

The table below is the normative entity catalog. Each entity carries a status: **New** (no TOGAF analogue), **Extended** (a TOGAF entity that gains attributes or semantics), or **Retained** (used as TOGAF defines it). Attributes and relationships are the ones the meta-model requires; an enterprise tailoring AEAF may add more but must not drop those marked here as load-bearing for the traceability chains in §5.5.

5.3.1 New and extended entities (the agentic additions)

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS
Actor	Extended	Anything that performs capabilities and bears responsibilities. No longer implies a human. Abstract; never instantiated directly.	id, name, type, accountabilityOwner	specializes to Human / Agent / Hybrid; performs Capability via Capability Allocation; answerable via Accountability Owner
Human Actor	Extended	A person, or an organizational role filled by people, performing work.	role, skills, authority	member of Hybrid Team; may be an Accountability Owner; performs Capability

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS
Agent Actor	New	A model-driven actor that perceives, decides, and acts toward an objective with limited human direction.	objective, autonomyLevel, permittedActions, lifecycleStatus	runs Model; draws on Knowledge Corpus; bounded by Guardrail; performs Action; assured by Eval Suite; granted an Autonomy Level; answerable via Accountability Owner; hosted by Application Component
Hybrid Team	New	A unit of human and agent actors performing a capability together with shared context and defined handoffs.	humanMembers, agentMembers, sharedContext, handoffRules	realizes Capability Allocation; governed by Oversight Control
Model	New	The underlying ML/LLM (or ensemble) that supplies an agent’s reasoning. Distinct from the agent that wraps it; one model can power many agents.	provider, version, modality, contextWindow, lifecycleStatus, costProfile	run by Agent Actor; evaluated by Eval Suite; catalogued in the Model Portfolio; hosted by Technology Component

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS
Knowledge Corpus	New	Data curated as a substrate for machine cognition — RAG corpora, embeddings/ vector indexes, memory stores, knowledge graphs.	sources, freshness, classification, embeddingMode l, accessPolicy	drawn on by Agent Actor; derived from Data Entity; classified for governance
Capability Allocation	New	The record of who or what performs a capability and under what oversight and autonomy. The central human–agent join.	performer, oversightMode, autonomyLevel, confidence, effectiveFrom/To	links Capability ↔ Actor; governed by Oversight Control; set at an Autonomy Level; rolls up to Automation Frontier
Automation Frontier	New	The derived, time-versioned boundary between human-performed and agent-performed work across the capability map.	asOfDate, coverage, trend	a view over Capability Allocations
Autonomy Level	New	A graded classification of how much independent action an agent is granted, from suggest-only to autonomous-within-bounds.	level, definition, requiredControls	attribute of Agent Actor and Capability Allocation; sets the Oversight mode

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS
Guardrail	New	A machine-enforceable constraint on agent behaviour — input filter, output check, tool permission, spend limit, action veto — enforced at runtime regardless of the agent's reasoning.	enforcementPoint, scope, condition, failAction, owner	bounds Agent Actor; constrains Action; realizes Principle; bounds Intent; feeds Assurance Case
Intent	New	The bounded statement of what outcome is to be achieved and within what limits — replaces the fully-specified requirement for non-deterministic work.	outcome, bounds, priority, acceptanceSignal	measured by Eval Suite; bounded by Guardrail; traces to Driver/Goal/Objective
Eval Suite	New	The maintained set of cases, probes, and metrics that bound and monitor a model's or agent's behaviour, replacing one-time verification.	dimensions (quality/safety/alignment/cost), thresholds, datasets, cadence, gate	assures Agent Actor; evaluates Model; measures Intent; feeds Assurance Case

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS
Assurance Case	New	The structured argument and evidence that an agent is fit to operate within its bounds.	claim, evidence, residualRisk, reviewDate	aggregates Eval Suite results and Guardrail logs for an Agent Actor; reviewed on a cadence
Oversight Control	New	How humans supervise an allocation: the mode and the escalation path.	mode (in/on/out-of-the-loop), trigger, escalationPath, interventionRights	governs Capability Allocation and Hybrid Team
Action / Tool	New	A concrete capability to affect the world that an agent may invoke (API call, transaction, message), with permissions and reversibility.	toolBinding, permission, rateBudget, reversibility, audit	performed by Agent Actor; constrained by Guardrail
Accountability Owner	New	The single named human or legal entity answerable for an agent's actions and outcomes. Accountability never transfers to an agent.	ownerRole, scope, since	anchors every Agent Actor and every high-consequence Action; exactly one per capability

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS
Principle	Extended	A normative architecture statement, now dual-audience, carrying an Enforcement attribute: advisory / design-time / runtime.	name, statement, rationale, implications, enforcement	compiled into Guardrail when enforcement = runtime

5.3.2 Retained TOGAF entities and how they connect

These entities keep their TOGAF meaning. Where AEAf adds an attribute or a relationship, the change is noted; otherwise they are used as-is. Their presence here is deliberate: AEAf extends the content metamodel, it does not replace it.

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS (AEAf ADDITIONS IN BOLD)
Role	Extended	A named set of responsibilities, as in TOGAF — but a Role may now be filled by a human, an agent, or a hybrid team.	name, responsibilities, permittedPerformerTypes	related to Actor via Capability Allocation
Function	Retained	A unit of business behaviour, independent of organizational structure.	name, description	performed by Actor; supported by Application Component

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS (AEAF ADDITIONS IN BOLD)
Process	Retained	An ordered set of activities that produces an outcome.	name, steps, trigger	performed by Actor; realizes Function
Service	Retained	A logical unit of behaviour with a defined interface (business or technology service).	name, interface, SLA	realizes Function; provided by Application Component
Application Component	Extended	An encapsulated software capability — as in TOGAF — but an AI service or agent runtime hosts the Agent it runs; the Agent is modelled separately.	name, type, hostsAgents	hosts Agent Actor; realizes Service
Data Entity	Extended	An encapsulated, recognizable thing of data — as in TOGAF — that may now be curated into a Knowledge Corpus for cognition.	name, owner, classification, feedsCorpus	source of Knowledge Corpus; manipulated by Process

ENTITY	STATUS	DEFINITION	KEY ATTRIBUTES	KEY RELATIONSHIPS (AEAF ADDITIONS IN BOLD)
Technology Component	Extended	A specific technology product or platform — as in TOGAF — now including inference platforms, vector stores, orchestration/runtime, and model gateways.	name, version, vendor	hosts Model and Knowledge Corpus; realizes Service
Driver / Goal / Objective	Retained	The motivation hierarchy: an external/internal condition (Driver), an intended end state (Goal), and a measurable target (Objective).	name, measure	traced by Intent
Building Block (ABB / SBB)	Retained	A reusable component of capability — an Architecture Building Block (logical) or Solution Building Block (implementation). AEAF adds agentic reference building blocks.	name, type, specification	composes the architecture; an SBB may be an Application Component that hosts an Agent

The combined catalog — §5.3.1 plus §5.3.2 — is the complete AEAF content meta-model. Two reading aids follow: the three structural moves that explain *why* the additions are shaped as they are (§5.4), and the traceability chains that explain *what the schema must hold together* (§5.5).

5.4 The three structural moves

Most of the new entities follow from three decisions about structure. Understanding the three moves is worth more than memorizing the catalog, because every downstream artifact and method phase is a consequence of them.

5.4.1 Move 1 — Actor is de-anthropomorphized

In the TOGAF content metamodel, "Actor" is, in practice, a person or organization unit. The whole discipline assumed it: only humans exercised judgment and initiative, so "actor performs function" needed no further qualification. The agentic enterprise breaks that assumption (→ see AEAF Framework §3, break A1). An agent perceives, decides, and acts toward an objective. It occupies the same conceptual slot a human used to own.

AEAF therefore makes **Actor abstract** and specializes it into **Human Actor**, **Agent Actor**, and **Hybrid Team**. The consequence is mechanical and pervasive: every relationship that used to read "an actor performs this function" now forces three questions the old schema could not even pose — *which kind of actor*, at *what autonomy level*, under *what oversight*. Those three questions are exactly the attributes Capability Allocation carries (§5.3.1).

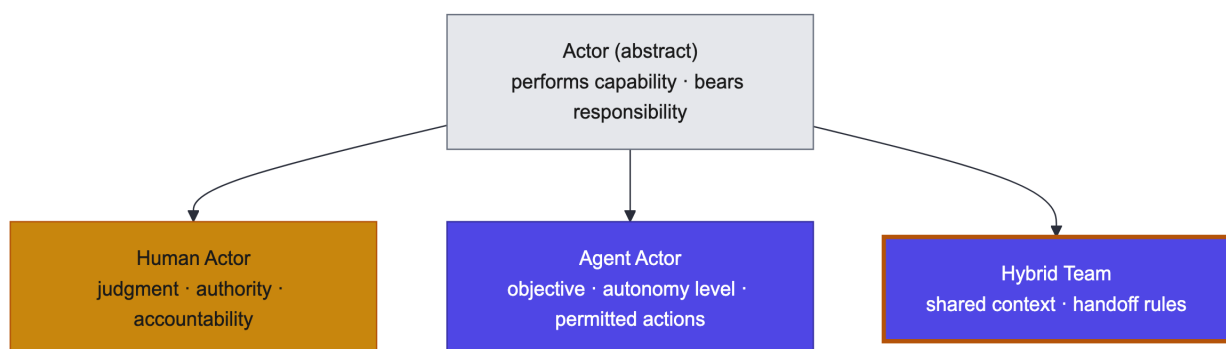


Figure 5.2 — The de-anthropomorphized Actor. The abstract Actor specializes into three concrete kinds. Amber is human judgment, indigo is agent intelligence, and the hybrid team carries both colours because it is jointly accountable for one capability. This single change is what lets a capability map describe a workforce that is part human, part agent.

The retained entities adapt without being rewritten: a Role gains `permittedPerformerTypes` so a role catalog can say "this role may be filled by an agent at autonomy L2 or by a human," and a Function is performed by an Actor of any kind. The schema becomes performer-agnostic by design — which is precisely what allows the same capability to be reallocated between humans and agents over time (§5.5, Chain C).

5.4.2 Move 2 — Model ≠ Agent ≠ Application

The most common modelling error in early agentic enterprises is to draw the agent as a box in the application layer — "the AI" as one more application component. AEAF separates three entities that the old schema would collapse into one, because lifecycle, evaluation, and governance attach at three different levels.

ENTITY	WHAT IT IS	WHAT ATTACHES TO IT	WHY IT MUST BE SEPARATE
Model	The reasoning engine — a large language or multimodal model, possibly an ensemble.	Provider, version, cost, evaluation against capability benchmarks; the Model Portfolio.	One model powers many agents; you swap, version, and price the model independently of any agent that uses it.
Agent Actor	The model placed in a loop with an objective, tools, memory, and guardrails.	Autonomy level, permitted actions, oversight, accountability owner, assurance case; the Agent Catalog.	The same model becomes a low-risk drafting assistant in one agent and a money-moving actor in another. Governance attaches to the agent, not the model.

ENTITY	WHAT IT IS	WHAT ATTACHES TO IT	WHY IT MUST BE SEPARATE
Application Component	The runtime that hosts and serves the agent — the orchestrator, the service, the deployment.	Interfaces, SLAs, scaling, the application portfolio.	A runtime can host several agents; replacing the runtime should not erase the agents' identity, history, or accountability.

The relationships encode the separation cleanly: an **Agent Actor** **runs** a **Model**, and an **Application Component** **hosts** an **Agent Actor**. Read in plain language: the Model is the capacity to reason; the Agent is the actor that reasons toward an objective with tools and bounds; the Application is the box that keeps the Agent running. A governance question — "is this agent permitted to issue refunds, and who answers if it errs?" — has nowhere sensible to live if the three are one box, and a clear home (the Agent Actor's **permittedActions**, its Guardrails, its Accountability Owner) once they are separated.

5.4.3 Move 3 — Principle compiles into Guardrail

Classical principles are advice to humans: an architect reads "customer PII is never sent to third parties" and designs accordingly. In the agentic enterprise the same sentence has a second reader — an agent acting at runtime (→ see AEA Framework §3, break A2). Advice an agent cannot read is not a control.

AEA gives every **Principle** an **Enforcement** attribute with three values, and adds a relationship: a principle whose enforcement is **runtime** is **compiled into** one or more **Guardrails** — its machine-enforceable form.

ENFORCEMENT VALUE	READER	MECHANISM	META-MODEL EFFECT
Advisory	Human	Read and applied as guidance, as in TOGAF.	Principle entity only; no Guardrail required.
Design-time	Human	Checked at an architecture gate / review.	Principle linked to a gate check; no runtime Guardrail.
Runtime	Agent	Enforced continuously on agent behaviour.	Principle compiled into Guardrail(s) with an enforcement point and a fail action.

"Compiled into" is the load-bearing verb. It means the Guardrail is not an independent invention but the derivative of a stated principle, so the chain Principle → Guardrail → enforcement point (§5.5, Chain B) stays navigable: an auditor can ask of any runtime guardrail "which principle does this enforce?" and of any runtime principle "where is it actually enforced?". A principle marked `runtime` with no Guardrail is a defect the schema can detect; a Guardrail tracing to no principle is an ungoverned constraint the schema can flag. The full catalog of AEAF principles and their enforcement values is Chapter 8; the practical compilation path is Book 2. → see AEAF Framework §8; AEAF in Practice §9.

5.5 The traceability the meta-model must preserve

The meta-model earns its place by holding four chains together. Each chain answers a question the agentic enterprise must always be able to answer, and each is the structural form of one promise from §5.1. If a chain cannot be navigated end to end in the repository, the corresponding governance claim is unsupported.

CHAIN	QUESTION IT ANSWERS	ENTITIES TRAVERSED
A — Intent assurance	Why did we build this, and how do we know it works?	Driver/Goal/Objective → Intent → Eval Suite → Assurance Case
B — Principle enforcement	What do we believe, and what is the agent actually constrained by?	Principle (runtime) → Guardrail → enforcement point
C — Allocation accountability	What work, performed by whom or what, under what oversight, and who answers?	Capability → Capability Allocation → {Actor, Oversight Control, Autonomy Level} → Accountability Owner
D — Continuous governance loop	How is behaviour bounded, watched, and corrected over time?	Model/Agent → Eval Suite → Assurance Case → Runtime/ Incident → corrective change (re-eval)

Chain D is the one that distinguishes AEAF most sharply from a gate-only practice: it is a *loop*, not a line. An incident in production feeds back to the eval suite (a new failure case) and to the assurance case (revised residual risk), which may trigger a change in guardrails, autonomy level, or allocation — and then re-enters evaluation. Figure 5.3 traces Chain D, the continuous-governance loop, end to end.

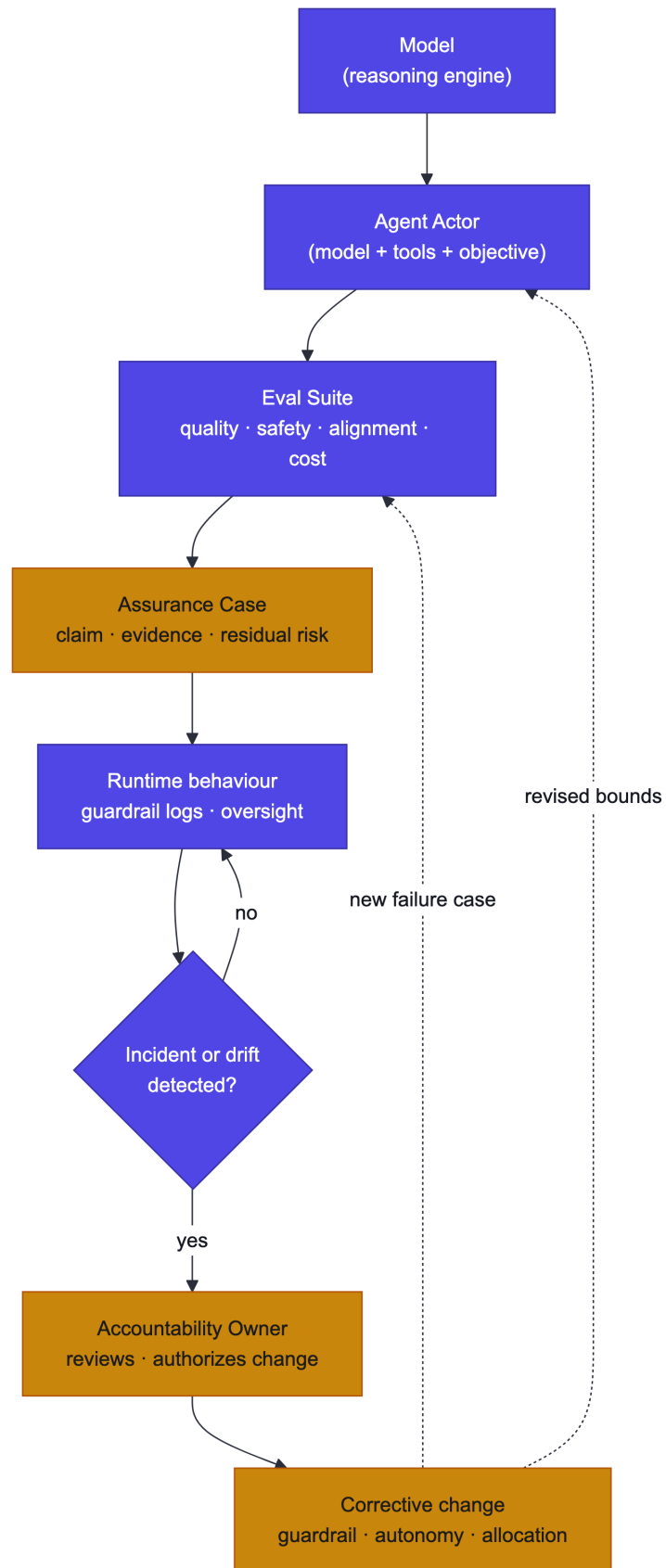


Figure 5.3 — Chain D, the continuous-governance loop. A model powers an agent; the agent is bounded and measured by an eval suite; the assurance case states the standing claim that the agent is fit to operate. Runtime behaviour is watched; a detected incident or drift escalates to the amber accountability owner, who authorizes a corrective change that re-enters evaluation. The dotted edges close the loop — this is governance that runs while the system runs, not a one-time sign-off.

Two properties of the chains are worth stating explicitly. First, every chain terminates — directly or through Capability Allocation — at an **Accountability Owner**: there is no path in the schema by which an agent's action escapes a named human answerability. Second, the chains share entities (Eval Suite appears in A and D; Guardrail in B and D; Capability Allocation in C and the Automation Frontier view), so a single repository, correctly schematized, supports all four queries without duplication. These chains are the backbone of Book 2's traceability spine. → see AEAf in Practice §18.

5.6 How the meta-model is used

The meta-model is not a diagram to admire; it is the schema the architecture repository implements and the source of structure for every AEAf artifact. Three uses follow directly.

As the repository schema. An AEAf-conformant repository stores instances of these entity types and enforces the required relationships. Conformance is checkable mechanically: a repository that lets you record an Agent Actor with no Accountability Owner, or a runtime Principle with no compiled Guardrail, is not AEAf-conformant, because it cannot hold the chains in §5.5. The entity catalog (§5.3) is the conformance specification.

As the schema behind the nine new artifacts. Each AEAf artifact is a *view* over a defined slice of the meta-model — a projection of certain entities and relationships into a catalog or matrix. The artifacts are specified in Chapter 21 and templated with worked examples in Book 2 Chapter 15; the table below names the entities each one projects, so the link from schema to artifact is explicit.

ARTIFACT (→ CH.21 / BOOK 2 CH.15)	PRIMARY META-MODEL ENTITIES IT PROJECTS
Agent Catalog	Agent Actor (+ Model, Action, Guardrail, Autonomy Level, Accountability Owner)
Model Portfolio	Model (+ the Agent Actors each powers, Eval Suite results)

ARTIFACT (→ CH.21 / BOOK 2 CH.15)	PRIMARY META-MODEL ENTITIES IT PROJECTS
Knowledge-Corpus Catalog	Knowledge Corpus (+ source Data Entity, the Agent Actors that retrieve from it)
Guardrail/Policy Catalog	Guardrail (+ the Principle each is compiled from, enforcement point)
Human-Agent RACI	Capability Allocation across Human/Agent/Hybrid (+ Accountability Owner)
Autonomy-Level Classification	Autonomy Level (+ each Capability Allocation's assigned level)
Eval-Suite Specification	Eval Suite (+ the Agent Actor / Model it assures, thresholds, gate)
Trust & Accountability Matrix	Capability → Accountability Owner, Oversight Control, Assurance Case
Capability Automation-Frontier Map	Automation Frontier (the view over Capability Allocations)

As the contract between the books. Because both volumes use the same locked entities, a term and a relationship mean the same thing in the framework and in the practice guide. Book 2's traceability matrix maps every playbook, template, and checklist back to the framework elements it realizes — and those elements are, at bottom, the meta-model entities defined here. The meta-model is therefore the shared vocabulary that makes "what affects what and why" answerable across both books. → see AEAf Framework §6 for how these entities distribute across the BDAT+I domains; → see AEAf in Practice §15, §18.

The meta-model defines the schema; it does not by itself populate it. Chapter 6 places these entities into the five BDAT+I domains. Chapter 7 develops the Actor model and the blended workforce. Chapter 8 supplies the principle catalog whose runtime members compile into the guardrails defined here. Chapter 9 begins the method that fills the repository this schema describes.

Chapter 6 — Architecture domains: BDAT + I

In brief

Enterprise architecture is organized into domains so that a complex enterprise can be described, owned, and changed in coherent parts. The classical four — Business, Data, Application, Technology (BDAT) — partition the enterprise cleanly when its actors are people and its systems are deterministic. The agentic enterprise introduces a population that BDAT has nowhere to put: the models, the agents that wrap them, the orchestration between agents, the knowledge curated for cognition, the guardrails that bound behaviour, the eval suites that assure it, and the autonomy levels and oversight that govern it. Bolting this onto Application or Technology forces three distinct concerns — *what reasons*, *what acts*, and *what hosts it* — into one box, and it loses the things that have no host at all (autonomy, guardrails, assurance). AEAF adds a fifth domain, **Intelligence & Agency** (the +I in BDAT+I), with a precise charter: it owns machine cognition and agency, and it explicitly does not own the transactional data, the runtime, or the business capability that remain with their classical domains. This chapter is the map of the five domains and how they interact. The per-domain method — how you actually do the work in each — is deferred to Chapters 11–15.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

6.1 Why architecture is organized into domains

A domain is a coherent area of architectural concern with its own entities, its own owners, and its own standards. Domains exist for three working reasons, not for taxonomy's sake.

REASON	WHAT THE DOMAIN PARTITION BUYS	FAILURE IF IT IS MISSING
Separation of concern	Each domain can be reasoned about, designed, and governed by people who specialize in it.	One team is asked to own everything; nothing is owned well.
Clear ownership	Every entity has a home domain that is accountable for its definition, lifecycle, and standards.	Orphan entities drift; no one is answerable for them.
Managed dependency	Cross-domain relationships are made explicit, so a change in one domain has a visible blast radius in the others.	Hidden coupling; a change in one place breaks something no one connected to it.

The test of a good domain set is simple: every entity the enterprise must describe has exactly one home, and every cross-domain dependency is visible at the boundary. AEAF's claim in this chapter is that BDAT fails the first half of that test for the agentic enterprise — there are entities with no home — and that adding one domain restores it.

The classical four (BDAT) — what each owns

TOGAF's four domains partition the enterprise along a stable, well-understood seam. AEAF retains all four; it does not redraw their internal boundaries.

DOMAIN	OWNS	REPRESENTATIVE ENTITIES
Business	What the enterprise does and why — its capabilities, value streams, processes, the actors and roles that perform work, and the organizational structure.	Capability, Value Stream, Process, Function, Actor, Role, Organization Unit, Driver, Goal.
Data	The information the enterprise holds and moves — its meaning, structure, quality, lineage, and governance.	Data Entity, Logical/Physical Data Component, Data Flow, master data, classification.

DOMAIN	OWNS	REPRESENTATIVE ENTITIES
Application	The software that supports the business — the application portfolio, the services applications expose, and the integration between them.	Application Component, Application Service, interface, integration.
Technology	The platform the software runs on — compute, storage, network, middleware, and the security infrastructure beneath it.	Technology Component, Platform Service, network, infrastructure security.

This partition holds because of the assumption stack named in the founding thesis (→ see AEAF Framework §3): actors are human, architecture is read by people, systems are deterministic, and change is gated. When those hold, *what reasons* and *what decides* live in the Business domain (people), and software is a deterministic support layer split cleanly into Application (the logic) and Technology (the platform). The agentic enterprise breaks all four assumptions at once, and the first crack to appear is in the domain partition.

6.2 Why BDAT has no home for machine intelligence and agency

Place the agentic population against BDAT and the misfit is not subtle. Seven kinds of thing need a home, and none of the four domains is the right one.

AGENTIC ENTITY	WHY IT DOES NOT FIT BDAT
Model	Not a Data Entity (it is not the enterprise’s information); not an Application Component (it is a reasoning engine that many applications and agents share); not pure Technology (its behaviour, cost, and risk profile must be governed, not just hosted).

AGENTIC ENTITY	WHY IT DOES NOT FIT BDAT
Agent	An actor that perceives, decides, and acts — but the Business domain’s actors were assumed human, and an Application Component is a passive piece of software, not an actor with an objective and an autonomy level.
Agent orchestration	The coordination of multiple agents toward an outcome is neither a business process (its performers are non-human and it changes at machine speed) nor an integration pattern (it routes cognition and initiative, not just messages).
Knowledge corpus	Data curated for machine cognition — embeddings, vector indexes, memory, graphs — is derived from Data but is not the transactional data itself; it is a cognitive substrate with its own freshness, classification, and access semantics.
Guardrail	A machine-enforceable constraint on agent behaviour has no BDAT home at all: it is not data, not an application, not infrastructure. It is a runtime control over an actor.
Eval suite	The continuing measurement that bounds a probabilistic actor replaces one-time testing. Testing was an Application/Technology activity against deterministic output; assurance is an ongoing governance object with no classical slot.
Autonomy level & oversight	How much independent action an agent is granted, and how a human supervises it, is a first-class design decision. BDAT has no entity for “how much may this actor decide on its own, and who watches?”

Why bolting +I onto Application fails

The tempting move is to call every agent an application and file the rest under it. It fails on three counts.

- **It collapses three distinct things into one box.** The Model (what reasons), the Agent (what acts toward an objective), and the runtime that hosts them are governed at different levels with different lifecycles — a point the meta-model makes structurally with `Model ≠ Agent ≠ Application` (→ see AEAF Framework §5). One model powers many agents; one runtime hosts many of each. Folding them into "an application" erases the relationships that governance needs.
- **It mis-types an actor as a feature.** An application is a passive support component; an agent is an actor that holds an objective, an autonomy level, and an accountability owner. Treating an actor as a feature is the central modelling error AEAF exists to correct (→ see AEAF Framework §3).
- **It still leaves entities homeless.** Even if you call agents applications, guardrails, eval suites, autonomy levels, and oversight controls have no place in the Application domain. They are not software components; they are controls *over* an actor's behaviour.

Why bolting +I onto Technology fails

The second tempting move is to treat all of it as "AI infrastructure" under Technology. This is closer — the inference platform, model gateway, vector store, and agent runtime *are* genuinely technology — but it confuses the host with the inhabitant.

- **Technology hosts; it does not reason or decide.** A vector store is a technology component; the knowledge corpus that lives in it is a cognitive substrate with its own governance. An inference platform is technology; the model it serves is a governed reasoning capability. The Technology domain answers *where does it run*, never *what does it decide and within what bounds*.
- **Governance objects are not platform objects.** Autonomy levels, guardrails, assurance cases, and oversight modes describe behaviour and accountability, not capacity and uptime. Filing them under Technology buries the enterprise's most consequential new risk surface inside an infrastructure catalogue no one reviews for behaviour.

The conclusion is structural, not stylistic: machine intelligence and agency form a coherent area of concern with its own entities, owners, and standards — the definition of a domain. AEAF gives it one.

6.3 The fifth domain: Intelligence & Agency (the +I)

The **Intelligence & Agency domain** is the fifth AEAF architecture domain. It owns the enterprise's machine cognition and its non-human agency: the population of models and agents, how they are orchestrated, the knowledge they reason over, the constraints that bound them, the evidence that assures them, and the autonomy and oversight under which they act. It is a domain, not a layer of the Application domain — its abbreviation is **+I**, and it completes **BDAT+I**.

What the +I domain owns

OWNED ENTITY	DEFINITION (HOME-DOMAIN SENSE)
Model	The underlying machine-learning model (typically a large language or multimodal model) that provides reasoning and generation. One model can power many agents; the +I domain owns its portfolio, version, risk profile, and lifecycle — not the inference hardware.
Agent (agent actor)	A software actor that pursues an objective by reasoning over a model, acting through tools, and operating within guardrails at a defined autonomy level. The +I domain owns the agent as an actor; the Business domain owns the capability the agent performs.
Agent orchestration	The coordination of multiple agents (and their handoffs to humans) toward an outcome — routing of cognition and initiative, not just of messages.
Knowledge corpus (cognitive substrate)	The governed body of content an agent retrieves from to ground its reasoning — documents, embeddings, vector indexes, memory, graphs — with defined sources, freshness, and access controls. Curated from Data; owned here for cognition.

OWNED ENTITY	DEFINITION (HOME-DOMAIN SENSE)
Guardrail	A machine-enforceable constraint that bounds an agent’s behaviour at runtime — input filter, output check, tool permission, spend limit, action veto — enforced regardless of the agent’s reasoning.
Eval suite	The maintained collection of cases, probes, and thresholds that measure an agent’s or model’s behaviour against expected outcomes, replacing one-time verification with continuing assurance.
Autonomy level	The graded degree of independent action granted to an agent for a capability, from suggest-only through bounded action to full autonomy within guardrails.
Oversight (mode & escalation)	How a human relates to an agent’s actions — in-the-loop, on-the-loop, or out-of-the-loop — and the designed path by which the agent hands control back to a human.

These map directly onto the new metamodel entities (→ see AEAF Framework §5) and onto the AEAF artifacts whose primary home is +I: the Agent Catalog, Model Portfolio, Guardrail/Policy Catalog, Autonomy-Level Classification, and Eval-Suite Specification (→ see AEAF Framework §21).

What the +I domain explicitly does NOT own

A domain is defined as much by what it excludes as by what it owns. Three exclusions keep the +I domain from swallowing its neighbours.

THE +I DOMAIN DOES NOT OWN	IT STAYS IN	WHY THE BOUNDARY IS DRAWN HERE
Raw transactional data — orders, claims, ledgers, customer records	Data	The enterprise’s information is governed once, in the Data domain, for all consumers. The +I domain owns the corpus curated for cognition, not the source data; the split is detailed in §6.5.

THE +I DOMAIN DOES NOT OWN	IT STAYS IN	WHY THE BOUNDARY IS DRAWN HERE
Hosting and runtime — inference platform, model gateway, vector store, agent runtime, compute and network	Technology	Where a model runs and what serves a corpus is a platform concern. The +I domain owns the model and the corpus as governed objects; Technology owns the infrastructure beneath them.
Business capability and accountability — the outcome-named ability the work delivers, and the human answerable for it	Business	A capability is actor-agnostic by design, which is what lets it be reallocated between humans and agents. The agent that performs it lives in +I; the capability and its accountability owner live in Business.

The discipline is one sentence: **the +I domain owns the intelligence and the agency, never the data it reasons over, the platform it runs on, or the business outcome it serves.** Those three remain with Data, Technology, and Business respectively, and are reached across explicit boundaries.

6.4 How the five domains relate

The five domains are not a stack and not a strict hierarchy; they are coherent areas of concern joined by named dependencies. The +I domain sits among the four classical domains and draws on each of them: it curates knowledge *from* Data, runs *on* Technology, exposes agents that *perform* Business capabilities, and is bound by Principles authored across the architecture.

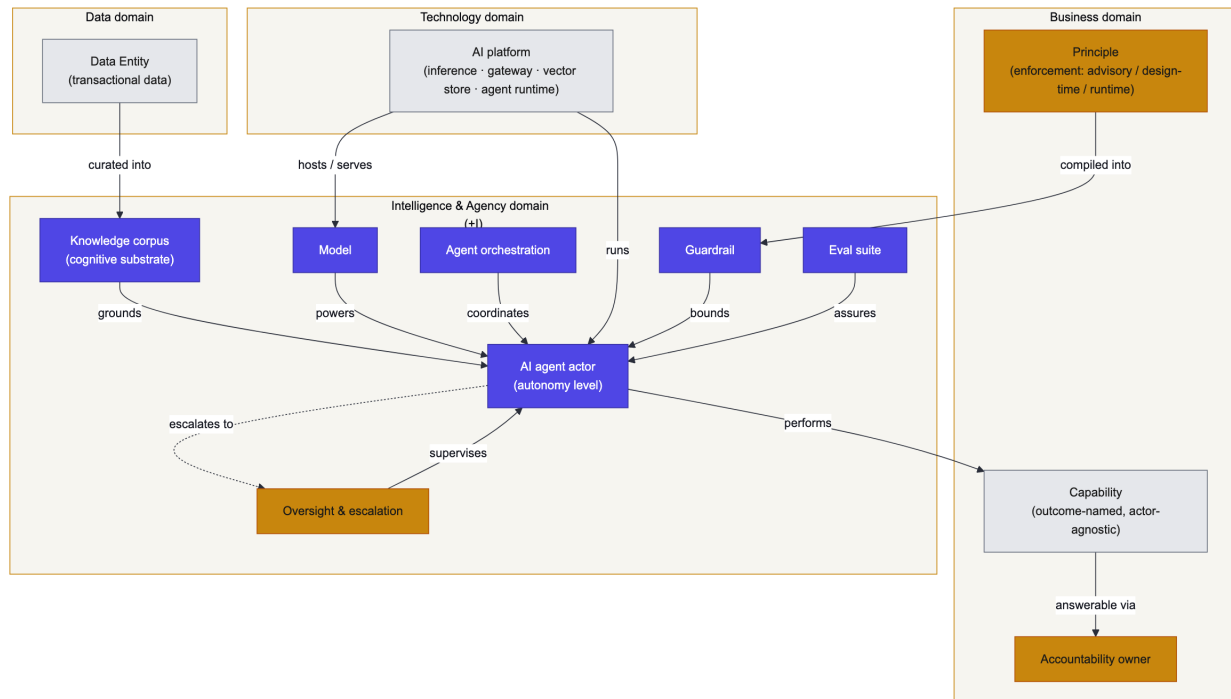


Figure 6.1 — The five AEAF domains and their cross-domain dependencies. Indigo nodes are machine intelligence (model, agent, orchestration, knowledge corpus, guardrail, eval); amber nodes are human judgment (accountability owner, principle author, oversight); grey nodes are classical BDAT entities. The load-bearing links: a knowledge corpus is *curated from* a Data entity and *grounds* an agent; a model *powers* the agent and is *hosted/served* by the Technology platform that also *runs* the agent; the agent *performs* a Business capability whose outcome is *answerable via* an accountability owner; a runtime-enforced principle is *compiled into* a guardrail that *bounds* the agent; an eval suite *assures* the agent; and the agent *escalates to* human oversight when it hits a bound. Every +I entity reaches into a classical domain across a named boundary — none of those boundaries is internal to +I.

The key cross-domain dependencies, read as sentences, are the contract between the domains:

- **Knowledge Corpus draws from Data.** The cognitive substrate is derived from transactional data but governed separately for freshness, classification, and access.
- **Agent is hosted on Technology.** The agent runtime, the model gateway, and the vector store are Technology components; the agent and model they carry are +I objects.
- **Agent performs a Business capability.** The agent is the performer; the capability is the work; the accountability owner — always human — answers for the outcome.
- **Guardrails realize Principles.** A principle with `enforcement = runtime` is compiled into one or more guardrails — the formal expression of architecture's second audience (→ see AEAF Framework §8).

- **Eval suites assure agents; oversight supervises them.** Assurance and oversight are the continuous-governance join between +I and the Business accountability it answers to.

6.5 Where Data splits: transactional data and the cognitive substrate

The cleanest new boundary in BDAT+I runs through what used to be one domain. Data now has two faces.

FACE OF DATA	HOME	WHAT IT IS	GOVERNED FOR
Transactional data	Data domain	The enterprise’s systems of record — orders, claims, ledgers, customer master data.	Accuracy, lineage, master-data discipline, regulatory retention.
Cognitive substrate	shared Data ↔ +I boundary	Data curated for machine cognition — RAG corpora, embeddings, vector indexes, agent memory, knowledge graphs.	Freshness, classification for retrieval, embedding-model lineage, access policy at query time.

The split matters because the same fact governed for a transaction is governed differently when an agent retrieves it to reason. A customer record in the Data domain is a row with an owner and a retention rule; the same record embedded in a knowledge corpus is a retrievable fragment whose freshness, classification, and access policy bound what an agent may ground a decision on. The Data domain owns the source; the +I domain owns the corpus; the Knowledge-Corpus Catalog records the derivation across the boundary. This chapter only marks the seam — the full treatment of the cognitive substrate, including embedding lineage and retrieval governance, is Chapter 12 (→ see AEAF Framework §12).

6.6 The five domains at a glance

Table 6.1 is the chapter's reference map: what each domain owns, its key entities, what the agentic enterprise changes in it, and where it touches the other domains. It is read as the charter of the five domains; the per-domain method is Chapters 11–15.

Table 6.1 — The BDAT+I domains: ownership, entities, agentic change, and interfaces.

DOMAIN	OWNS	KEY ENTITIES	WHAT CHANGES IN THE AGENTIC ENTERPRISE	INTERFACES TO OTHER DOMAINS
Business	What the enterprise does and why; who or what performs it and who is accountable.	Capability, Value Stream, Process, Actor, Role, Capability Allocation, Accountability Owner, Principle.	Capability becomes actor-agnostic and is allocated to human / agent / hybrid; the automation frontier is mapped; principles gain an Enforcement attribute.	→ +I: agents perform capabilities; principles compile into guardrails. → Data: capabilities consume data.
Data	The enterprise's transactional information — meaning, structure, quality, lineage, governance.	Data Entity, Data Flow, master data, classification.	Source data also feeds the cognitive substrate; classification now governs what may be retrieved into a corpus.	→ +I: data is curated into knowledge corpora. → Application/Technology: as classically.

DOMAIN	OWNS	KEY ENTITIES	WHAT CHANGES IN THE AGENTIC ENTERPRISE	INTERFACES TO OTHER DOMAINS
Application	The software portfolio and the integration between applications.	Application Component, Application Service, interface.	Applications must tolerate non-determinism: bounded outputs, fallback paths, human handoff, observability of probabilistic components.	→ +I : an application may invoke an agent; an agent runtime is an Application/Technology component while the agent it hosts is a +I object.
Technology	The platform beneath the software — compute, storage, network, security, and now the AI platform.	Technology Component, Platform Service, inference platform, model gateway, vector store, agent runtime.	Adds the AI platform / inference / LLMOps substrate as a first-class technology concern.	→ +I : hosts models, serves corpora, runs agents. The platform never owns the behaviour it carries.
Intelligence & Agency (+I)	Machine cognition and non-human agency: models, agents, orchestration, knowledge corpora, guardrails, evals, autonomy levels, oversight.	Model, Agent, Agent Orchestration, Knowledge Corpus, Guardrail, Eval Suite, Autonomy Level, Oversight Control.	The whole domain is new — it gives machine intelligence and agency a first-class home with its own entities, owners, and standards.	→ Business : agents perform capabilities, bounded by compiled principles, answerable to owners. → Data : corpora curated from data. → Technology : hosted and run on the AI platform.

6.7 What this chapter fixes, and what comes next

BDAT partitions the enterprise cleanly for human actors and deterministic systems. It has no home for models, agents, orchestration, knowledge curated for cognition, guardrails, evals, autonomy, and oversight — and bolting these onto Application collapses distinct things into one box while leaving controls homeless, while bolting them onto Technology confuses the host with what it hosts. AEAF adds one domain, **Intelligence & Agency (+I)**, with a charter precise enough to keep it from swallowing its neighbours: it owns the intelligence and the agency, and it leaves transactional data with Data, hosting with Technology, and business capability and accountability with Business. The five domains relate through named cross-domain dependencies (Figure 6.1), and the Data domain itself splits at the cognitive-substrate seam handled in Chapter 12.

This chapter is the map. The territory — how to actually do the work in each domain — follows: business and capability architecture in Chapter 11, data and knowledge in Chapter 12, application in a non-deterministic world in Chapter 13, technology and the AI platform in Chapter 14, and the new Intelligence & Agency architecture in Chapter 15.

Chapter 7 — The reworked Actor model & the blended workforce

In brief

When a role can be filled by a person, an AI agent, or both, the *who* of work stops being given and becomes a thing to design. This chapter takes the de-anthropomorphized Actor from the meta-model (→ see AEAF Framework §5) and turns it into an operating model: how capabilities are allocated across human and agent actors, how the **automation frontier** — the boundary between human-performed and agent-performed work — is mapped and moved, how **oversight modes** tie human involvement to consequence and confidence, and how **hybrid teams** share a task without dropping it between them. The unifying act is **capability allocation**: a first-class decision recording who or what performs a capability, at what autonomy level, under what oversight, and who stays accountable. A claims-refund capability runs through the chapter to show allocation, frontier, and oversight working together. The formal entities live in the meta-model; the deep failure modes of teaming are deferred to Book 2 Ch.19; the method phase that produces these decisions is Ch.11.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

1. The de-anthropomorphized Actor as the unit of organizational design

Classical enterprise architecture treats an **actor** as a person or an organizational role filled by people. That assumption is invisible because it was never false — until agents arrived. AEAF makes Actor abstract and specializes it into three concrete kinds, and this single move propagates through every org chart, role catalog, and capability map in the enterprise.

ACTOR KIND	WHAT IT IS	WHAT IT BRINGS	WHAT IT CANNOT DO
Human actor	A person, or an organizational role filled by people.	Judgment, accountability, handling of genuine novelty and exception, relationship and ethical weight.	Operate at machine speed or scale; stay tireless and consistent across thousands of cases.
Agent actor	An AI agent: a model in a loop with tools, memory, and an objective, bounded by guardrails and a defined autonomy level.	Speed, scale, around-the-clock consistency, tireless execution within bounds.	Hold accountability; reliably handle situations outside its evaluated envelope; be exhaustively specified.
Hybrid team	A unit of human and agent actors jointly performing one capability with shared context and defined handoff rules.	Coverage of the volume by agents and the exceptions by humans, in one workflow.	Function without explicit handoff, escalation, and a single accountability owner.

The consequence for organizational design is direct: **a role is no longer a synonym for a person**. A role in AEAF carries a `permittedPerformerTypes` attribute — it may be filled by a human, an agent, a hybrid team, or any of these over time. "Claims triage officer" becomes a role that *can* be performed by a human, by an agent, or by a hybrid team, and the architecture's job is to decide which, for which work, and under what oversight.

This forces three changes to how organizations are described.

- **Functions decouple from headcount.** A function ("settle low-value refunds") is sized by the work, not by how many people it would take, because some or all of it may be performed by agents. Capacity planning becomes capability-and-allocation planning.
- **Org structure gains a supervision layer.** Where work is agent-performed, the human role shifts from *doing* to *overseeing*, *handling exceptions*, and *owning outcomes*. A team of fifteen processors plus one supervisor can become three exception-handlers plus an accountability owner plus a fleet of agents — a different structure, not a smaller version of the old one.
- **Accountability is pinned, not distributed.** Because an agent actor cannot be accountable (→ see AEAF Framework §4), every capability that any agent touches must name exactly

one human **accountability owner**. De-anthropomorphizing the Actor does not de-anthropomorphize accountability.

Design rule. Model capabilities actor-agnostically; decide the performer per capability through allocation. Never bake "human" or "agent" into the capability name. "Settle a refund" is the capability; who settles it is an allocation decision that can change.

2. Capability allocation as a first-class decision

A **capability allocation** is the recorded decision of which actor performs a capability, at what autonomy level, under what oversight, and who is accountable. It is the central design act of the blended workforce. Allocation is not staffing and not resourcing — those answer *how many*; allocation answers *what kind of actor, with how much independence, watched how closely*.

2.1 The allocation question

Every allocation answers four sub-questions together. Answering one without the others produces an allocation that looks decided but is not governable.

1. **Performer** — human actor, agent actor, or hybrid team?
2. **Autonomy level** — how much independent action is the agent granted, from suggest-only through bounded action to full autonomy within guardrails? (The Autonomy-Level Classification is defined in its home chapter; this chapter uses it, not defines it.)
3. **Oversight mode** — in-the-loop, on-the-loop, or out-of-the-loop? (Section 4.)
4. **Accountability owner** — which single named human or role answers for the outcome, including the agent-performed part?

These four are interlocked. A high autonomy level with in-the-loop oversight is incoherent (you granted independence and then withheld it). Out-of-the-loop oversight on a high-consequence, low-confidence capability is reckless. The allocation framework exists to keep the four consistent.

2.2 The allocation factors

Six factors decide whether — and how far — a capability moves from human to agent performance. They are read together, not scored in isolation; one disqualifying factor (an irreversible, high-consequence action with immature evals) overrides several favourable ones.

FACTOR	QUESTION IT ANSWERS	PUSHES TOWARD AGENT / HIGHER AUTONOMY	PUSHES TOWARD HUMAN / LOWER AUTONOMY
Consequence	How much harm if the action is wrong?	Low: minor, recoverable, bounded cost.	High: safety, money at scale, legal exposure, irreversible customer harm.
Reversibility	Can the action be undone?	Easily reversed (re-issue, recompute, roll back).	Irreversible (payment sent, message to a regulator, data deleted).
Ambiguity	How clear-cut is the task?	Well-specified, narrow input space, clear right answer.	Open-ended, judgment-laden, novel situations common.
Confidence / eval maturity	How well do we know the agent behaves?	Mature eval suite, stable behaviour, low observed error rate.	Thin or no evals, drifting behaviour, unknown failure modes.
Regulatory constraint	Does law require a human decision?	No constraint, or human review preserved downstream.	A human decision-maker is legally mandated (e.g. certain credit, medical, or HR decisions).
Cost	What is the economics of each performer?	Volume high, per-case human cost dominant, agent marginal cost low.	Volume low, human cost trivial, agent build/assurance cost not justified.

Confidence and consequence are the dominant pair. The other four shape the decision, but the mode and autonomy granted are driven first by *how bad is wrong* against *how sure are we* — which is why Section 4's oversight table is built on exactly those two axes.

2.3 The allocation decision flow

The decision flow turns the factors into a repeatable path. It is deliberately conservative: the default for a consequential, uncertain action is human performance, and an agent earns autonomy by demonstrating evaluated, bounded behaviour — not by assertion.

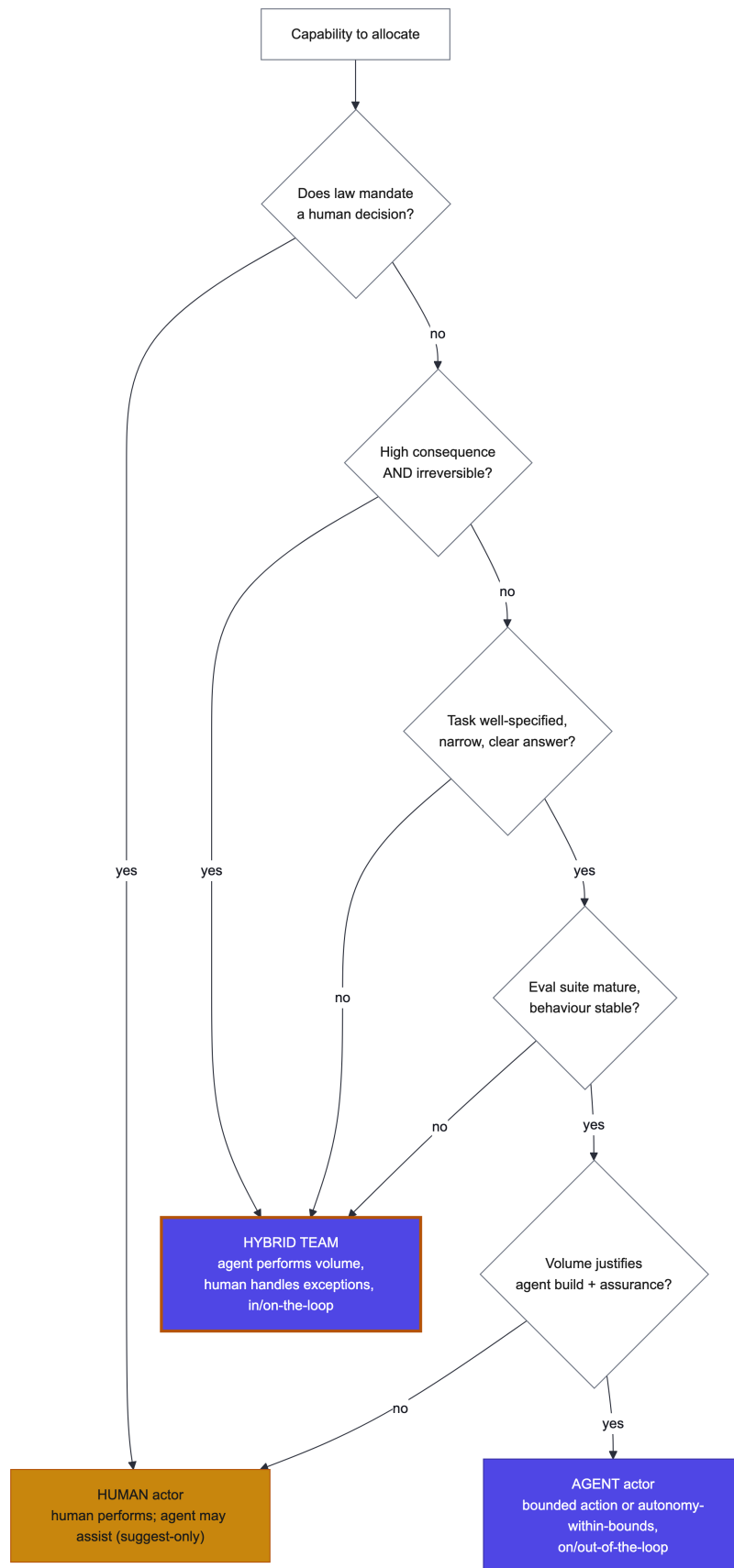


Figure 7.1 — The capability-allocation decision flow. The flow defaults consequential or uncertain work to humans (amber) and routes the rest through a hybrid team (split) before granting an agent (indigo) bounded or autonomous-within-bounds performance. A legal mandate for human decision short-circuits to human performance regardless of the other factors; thin evals route to a hybrid team so the agent runs under close human oversight while confidence is built. The flow is read top to bottom for a single capability; re-run it when any factor changes.

The flow's bias is intentional. It is far cheaper to grant an agent more autonomy later, once evals mature and behaviour proves stable, than to recover from an autonomous agent acting wrongly at scale on a high-consequence, irreversible action. Allocation starts conservative and is *promoted* with evidence.

3. The automation frontier

The **automation frontier** is the boundary, per capability, between work performed by human actors and work performed by agent actors — and the direction that boundary is expected to move. It is the aggregate view across all of an enterprise's capability allocations, and it is the single most important picture of where the blended workforce stands today and is heading.

3.1 The frontier is time-versioned, not fixed

A frontier is a position *as of a date*. Each capability allocation carries an `effectiveFrom` date, so the frontier can be stated for any point in time and compared across dates. This matters because the agentic enterprise's defining feature is that the boundary **moves**: capabilities cross from human to agent performance as agents become more capable and as trust is earned. Describing the frontier as a fixed split ("60% automated") loses the only information that makes it useful — its trajectory.

3.2 Mapping the frontier

The frontier is captured in the **Capability Automation-Frontier Map**, a view over the capability model that records, per capability, the current performer mix, the autonomy level, the oversight mode, and the planned next move. To map it:

1. **Take the capability model as the spine.** Capabilities are actor-agnostic, so they are the stable axis against which the human/agent boundary is plotted.

2. **Record the current allocation per capability** — performer, autonomy level, oversight mode, accountability owner.
3. **Mark the trajectory** — is this capability staying human, being piloted as hybrid, or planned to move further toward agent performance, and by when?
4. **Re-stamp the date.** The map is re-issued on a cadence (and on any material allocation change) so its movement is visible.

3.3 Why and how the frontier moves

The frontier advances for two distinct reasons, and confusing them is dangerous.

DRIVER OF A FRONTIER MOVE	WHAT CHANGED	WHAT IT JUSTIFIES
Capability improved	A better model, better tools, or a matured eval suite means the agent now handles work it could not before.	Allocating new kinds of work to the agent — a genuine extension of the frontier.
Trust earned	The agent has run under oversight with a low observed error rate; its assurance case is stronger.	Stepping down oversight or raising autonomy on work the agent already does — same work, looser supervision.

The frontier can also **retreat**: trust lost (a rising error rate, an incident, drift detected) or capability withdrawn (a model deprecated, a regulation tightened) moves work back toward human performance or tighter oversight. A frontier that only ever advances is not being governed — it is being assumed safe.

3.4 Planning frontier moves, and the risk of moving without assurance

A frontier move is a planned transition, not an event that happens to the organization. Planning a move means deciding the target allocation, the assurance evidence required to justify it, the oversight step-down sequence, and the human transition (the people whose work shifts from doing to overseeing or to other capabilities). This planning is the work of the method's opportunities-and-migration phase (→ see AEAF Framework §16) and Ch.11; this chapter establishes the frontier as the object that is planned.

The central risk is **moving the frontier without assurance** — granting an agent more autonomy or stepping down oversight because it is cheaper or faster, without the eval evidence that the agent behaves within bounds. The failure is silent: the agent keeps acting,

the cost saving is immediate and visible, and the wrong decisions accumulate out of sight until an incident surfaces them. AEAF's rule is that **the frontier moves on evidence, and the evidence is the assurance case** — evals, guardrail logs, and oversight history showing the agent behaves within its intent and bounds (→ see AEAF Framework §17). No assurance, no move.

4. Oversight modes

Human involvement in agent work is not binary. AEAF uses three named modes, and using them precisely is binding: "human-in-the-loop" is not a catch-all for *any* human involvement, it is one specific mode.

4.1 The three modes, operationally

MODE	WHAT IT MEANS OPERATIONALLY	HUMAN'S RUNTIME ROLE	LATENCY / SCALE EFFECT
In-the-loop	The agent proposes; a human approves each action before it executes. Nothing reaches the world unapproved.	Approver on every case. The human is a gate in the runtime path.	Slow; throughput capped by human review capacity. Highest control.
On-the-loop	The agent acts; a human monitors a stream of actions and can intervene , pause, or override. Control is by exception, not per case.	Supervisor of a fleet; intervenes when a signal or sample warrants.	Fast; throughput limited by agent, not human. Control via monitoring + intervention rights.
Out-of-the-loop	No human in the runtime path. The agent acts within guardrails; control is the guardrails plus after-the-fact review and evals.	None at runtime. Human owns the guardrails, the assurance case, and incident response.	Fastest, full scale. Control is entirely pre-set (guardrails) and retrospective (review).

A mode is not a maturity badge — out-of-the-loop is not "better" than in-the-loop. Each is the *correct* mode for a particular combination of consequence and confidence. The skill is matching the mode to the work.

4.2 Mode ties to autonomy level and consequence

The oversight mode and the agent's autonomy level are two views of the same grant. A suggest-only agent is in-the-loop by construction (its suggestions do nothing until a human acts). An autonomy-within-bounds agent must be on- or out-of-the-loop, because in-the-loop approval would negate the autonomy granted. The mode is set primarily by **consequence × confidence**: how bad it is when the agent is wrong, against how sure we are that it is right.

	LOW CONFIDENCE (THIN EVALS, NEW, DRIFTING)	MEDIUM CONFIDENCE (SOME EVALS, OBSERVED BUT LIMITED)	HIGH CONFIDENCE (MATURE EVALS, STABLE, LOW ERROR RATE)
High consequence (irreversible / large harm)	In-the-loop — human approves each action.	In-the-loop, stepping to on-the-loop for the well-evaluated subset.	On-the-loop — monitored, with intervention rights; in-the-loop retained for the highest- stakes slice.
Medium consequence (recoverable but material)	In-the-loop.	On-the-loop.	On-the-loop, moving toward out-of-the- loop within tight guardrails.
Low consequence (minor, easily reversed)	On-the-loop.	On-the-loop / out-of- the-loop.	Out-of-the-loop within guardrails, sampled review.

Table 7.1 — Consequence × confidence → oversight mode. Consequence is the dominant axis: high-consequence work stays close to a human even at high confidence, because the cost of a rare error is large. Confidence governs how far down the oversight ladder a given consequence band may step. Cells name the default mode and, where relevant, the direction of travel as confidence grows.

4.3 Mapping autonomy to mode

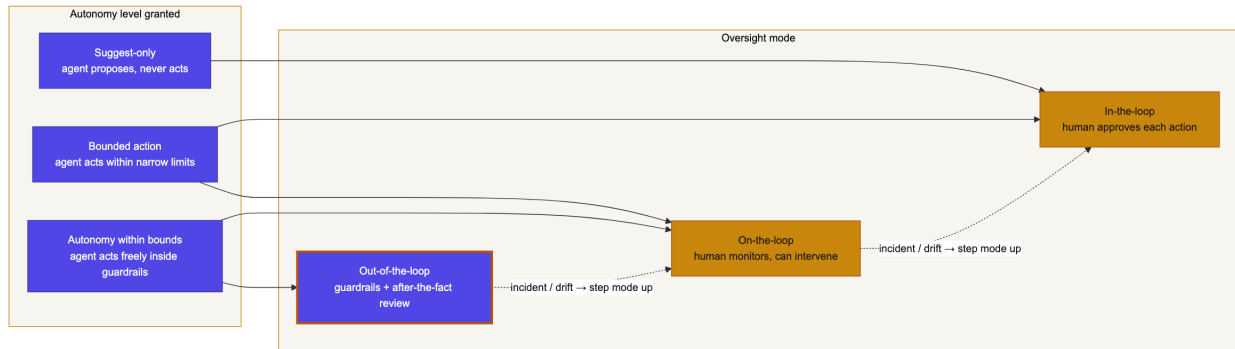


Figure 7.2 — Autonomy level mapped to oversight mode. A higher autonomy grant (indigo) pairs with a looser oversight mode (amber), but the mapping is many-to-one in the safe direction: a bounded-action agent may run in-the-loop *or* on-the-loop depending on consequence and confidence, never the reverse. The dotted edges are the step-up path — an incident or detected drift tightens oversight (out-of-the-loop → on-the-loop → in-the-loop) without necessarily revoking the autonomy grant itself.

4.4 Stepping modes up and down

Oversight is adjusted, not fixed at allocation time.

- **Step down** (looser oversight — in → on → out) when confidence rises: the eval suite matures, the observed error rate stays low under monitoring, and the assurance case supports it. Stepping down is a *planned, evidenced* move, the runtime twin of advancing the frontier (Section 3).
- **Step up** (tighter oversight — out → on → in) immediately on an incident, a detected behaviour drift, an eval threshold breach, or a change in consequence (a new high-value case class enters the capability). Stepping up does not wait for a governance meeting; it is a runtime control.

The step-up path must be designed *before* it is needed, because the moment to tighten oversight is exactly the moment when something has gone wrong and there is no time to invent the mechanism.

5. Hybrid teams

A **hybrid team** is a unit of human and agent actors jointly performing one capability with shared context and defined handoff rules. It is the most common real allocation, because most capabilities are neither "all human" nor "all agent" — agents take the volume, humans take the exceptions, and the two share the same work.

5.1 Sharing task and context

The defining property of a hybrid team is **shared context**: the human and the agent work on the same case, with the same information, without re-keying or re-explaining. When an agent escalates a refund to a human, the human sees what the agent saw, what it decided, and why it stopped — not a fresh ticket. When a human overrides an agent, the override and its reason re-enter the agent's context. A hybrid team without shared context is two separate actors with a queue between them, and it fails at exactly the handoff points where it is supposed to be strong.

5.2 Handoff and escalation patterns

PATTERN	DIRECTION	TRIGGER	WHAT TRANSFERS
Escalation	agent → human	The agent hits a bound, low confidence, an ambiguous case, or a guardrail veto.	Full case context, the agent's partial work, the reason for stopping.
Delegation	human → agent	A human assigns the agent a sub-task within a case the human owns.	The sub-task scope and its bounds.
Override	human → agent	A human corrects or reverses an agent's action or decision.	The corrected outcome and the reason, fed back into context.
Resumption	human → agent	The human resolves the escalation and returns the case for the agent to complete.	The human's resolution and any new constraint.

Escalation is the load-bearing pattern. An escalation is a *designed path*, not a failure — it specifies the trigger, the target human, and the fallback if no human is available (hold, queue, or safe-default action). An agent with no escalation path either acts beyond its competence or stalls silently; both are worse than a clean handoff.

5.3 The human factors of teaming

Sharing work with an agent introduces failure modes that are about the *human's relationship to the agent*, not the agent's behaviour. AEAF names them here so they are designed against; their full mechanics and the design countermeasures are deferred to Book 2 Ch.19.

- **Over-trust (automation bias).** The human accepts the agent's output without the scrutiny the work warrants, because it is usually right. Approval becomes a rubber stamp, and in-the-loop oversight quietly becomes out-of-the-loop without anyone deciding so.
- **Under-trust.** The human distrusts a competent agent and re-checks everything, erasing the throughput gain and defeating the allocation. Under-trust is as much a design failure as over-trust.
- **Automation complacency.** Sustained monitoring of a reliable agent erodes vigilance; the human stops genuinely watching the stream they are nominally supervising, so on-the-loop oversight degrades to nominal-only.

These are not edge cases — they are the predictable human response to working alongside a capable, mostly-right agent, and they determine whether an oversight mode delivers the control it claims on paper. Designing the hybrid team means designing against them (e.g. calibrated confidence display, sampled forced-review, rotating attention), which Book 2 Ch.19 treats in full.

6. Worked example: the refund-settlement capability

One capability carries the chapter's ideas together. It is illustrative — the figures are chosen to make the mechanics legible, not drawn from a specific organization.

The capability. *Settle a customer refund* — an e-commerce retailer receives refund requests, decides validity and amount, and issues the refund. Roughly 10,000 requests a month. The capability is actor-agnostic; the allocation decision is what kind of actor settles which requests.

Step 1 — Allocate using the factors (Section 2). The capability is not uniform. Decomposing it by the allocation factors reveals three bands with different allocations.

REQUEST BAND	CONSEQUENCE	REVERSIBILITY	AMBIGUITY	ALLOCATION
Standard refund ≤ EUR 200, clear policy match, item returned	Low	Reversible (re- debit possible)	Low — clear right answer	Agent actor, autonomy- within-bounds
Refund EUR 200–2,000, or partial / policy- edge cases	Medium	Reversible	Medium — judgment needed	Hybrid team — agent proposes, human confirms
Refund > EUR 2,000, fraud- flagged, or legal/ chargeback exposure	High	Effectively irreversible at scale	High — novel, contested	Human actor, agent assists (suggest-only)

A single accountability owner — the Refunds Operations Lead — answers for all three bands, including the agent-settled one.

Step 2 — Place the frontier (Section 3). As of the allocation date, the frontier sits inside the capability: the EUR 200 band is agent-performed, the mid band is hybrid, the high band is human. The trajectory, planned for the next two quarters, is to raise the agent's bound from EUR 200 toward EUR 500 *if* the eval suite on the standard band holds a low error rate under monitoring — a frontier move justified by trust earned, not by the cost saving alone. The Capability Automation-Frontier Map records this position and trajectory, dated.

Step 3 — Set oversight per band (Section 4, Table 7.1).

BAND	CONSEQUENCE × CONFIDENCE	OVERSIGHT MODE	AUTONOMY LEVEL
Standard ≤ EUR 200	Low consequence × high confidence	Out-of-the-loop within guardrails (spend cap, policy check, sampled review)	Autonomy within bounds

BAND	CONSEQUENCE × CONFIDENCE	OVERSIGHT MODE	AUTONOMY LEVEL
Mid EUR 200–2,000	Medium consequence × medium confidence	On-the-loop — agent proposes, human confirms by exception	Bounded action
High > EUR 2,000	High consequence × low confidence (novel cases)	In-the-loop — human decides, agent supplies analysis	Suggest-only

Step 4 — Run it as a hybrid team (Section 5). The agent settles the standard band autonomously, escalating any case that hits a guardrail (amount over cap, policy ambiguity, fraud signal) to the on-the-loop human with full context: what it saw, the policy it matched, why it stopped. The human resolves and, where the resolution reveals a new pattern, the override feeds back so the agent handles the next similar case. The accountability owner watches the over-trust risk on the mid band — if the human's confirm-rate climbs toward 100% without genuine review, on-the-loop oversight has silently become rubber-stamping, and the mode must be re-examined.

Step 5 — Move the frontier only on assurance. When the team proposes raising the agent's bound to EUR 500, the move is gated on the assurance case for the standard band: the eval suite's measured error rate, the guardrail-veto logs, and the override history. If the evidence holds, the bound is raised and oversight on the new sub-band starts at on-the-loop, stepping toward out-of-the-loop as confidence at the higher amount accumulates. If an incident or drift appears at any point, oversight steps back up immediately (Section 4.4). The frontier moved because the agent earned it — and it can retreat the same way.

This is the blended workforce as an operating model: one capability, one accountability owner, three allocations, three oversight modes, a frontier that sits *inside* the capability and moves on evidence — and a hybrid team that shares context across every handoff.

7. What this chapter hands forward

The Actor model reworked here is the spine the rest of Book 1 hangs allocation, oversight, and accountability on.

- The **formal entities** behind every term used here — Actor, Capability Allocation, Autonomy Level, Oversight Control, Accountability Owner — are defined in the meta-model (→ see AEAF Framework §5).
- The **method phase** that actually produces allocations and the automation-frontier map is business & capability architecture (→ see AEAF Framework §11), with frontier-move planning and the human transition in opportunities & migration (→ see AEAF Framework §16).
- The **RACI / decision-authority artifact** that records the human/agent split per capability is the Human-Agent RACI, catalogued in Ch.21 and templated in Book 2.
- The **assurance** that gates every frontier move and oversight step-down is Intent & Assurance management (→ see AEAF Framework §17).
- The **deep human factors** named in Section 5.3 — over-trust, under-trust, automation complacency — and the way to design against them are treated in full in Book 2 Ch.19.

Chapter 8 — Principles for the Agentic Enterprise

In brief. An architecture principle is a foundational rule that guides decisions. For the whole history of the discipline a principle had exactly one audience — the human who read it and chose to honour it. The agentic enterprise adds a second audience: the agent that must obey the principle while it acts, at machine speed, with no human in the moment. This is Ari's observation made formal, and it changes what a principle *is*. AEAF keeps TOGAF's principle structure — Name, Statement, Rationale, Implications — and adds one attribute that carries the whole weight of the change: **Enforcement**, valued *advisory*, *design-time*, or *runtime guardrail*. A runtime principle is no longer only prose; it is **compiled** into a machine-enforceable guardrail. This chapter explains the dual-audience principle, defines the Enforcement attribute and how to choose its value, shows the compilation path with a worked example, and sets out **the AEAF Principle Catalog**: eighteen principles for the agentic enterprise, grouped foundational / safety & trust / assurance / workforce / architecture.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

8.1 What a principle is for, and the structure AEAF keeps

A principle sits above standards, patterns, and designs. It is the durable statement of *what the enterprise believes is right*, from which more specific rules are derived and against which proposed changes are tested. A standard can be swapped; a pattern can be re-chosen; a principle is meant to outlast both. TOGAF formalised the principle into four parts, and they remain the right four. AEAF retains them exactly:

PART	WHAT IT CARRIES	EXAMPLE FRAGMENT
Name	A short memorable handle the organisation can refer to.	"Accountability is never delegated."

PART	WHAT IT CARRIES	EXAMPLE FRAGMENT
Statement	The rule itself, as a single normative sentence.	"Every agent action traces to a named human or legal entity who is answerable for it."
Rationale	Why the enterprise holds it — the reasoning that survives challenge.	"An agent has no legal standing and cannot bear consequences; someone must."
Implications	What honouring the principle forces — the costs and obligations it creates.	"Every capability has exactly one accountability owner recorded before an agent may act in it."

The structure works because it separates the *rule* from the *reason* from the *cost*. A reader who disputes the rule is sent to the rationale; a reader who accepts it is sent to the implications. AEAF does not improve on this. It extends it, because in the agentic enterprise the four parts are no longer the whole of what a principle must carry.

8.2 The dual-audience principle

Here is the change that drives this chapter. In classical EA a principle is **read by a human and applied by a human**. An architect reads "data is an asset," internalises it, and brings it to bear when designing. The principle never executes; it persuades. Its only enforcement is the discipline and judgment of the people who read it — and, at a design gate, the reviewer who checks the design against it.

In the agentic enterprise an agent reads architecture *at runtime*. A principle such as "customer personal data is never sent to a third-party model" is no longer only advice an engineer keeps in mind while building. It is a constraint that an autonomous actor must honour in the moment it decides which model to call — with no human in that moment to interpret it. The principle has acquired a **second audience**: the agent that must obey it while acting.

Ari's observation, stated as a framework rule: a principle in the agentic enterprise has two audiences — the human architect or engineer who reads it at design time, and the agent actor that must obey it at runtime. A principle that only the first audience can act on is, for the second, merely decoration.

The consequence is concrete and unforgiving. A human reader can interpret "minimise data exposure" with judgment; an agent cannot act on a sentence it can only read. For the second audience the principle has to exist in a form the runtime can *check and enforce* — a condition, a permission, a limit, a veto — not a paragraph of guidance. So a principle that is meant to bind agent behaviour must be expressible twice: once as the human statement (which still persuades and explains), and once as the machine-enforceable form (which actually binds). This dual expression is the default in AEAF, not an afterthought — it is design tenet 2 of the framework (→ see AEAF Framework §3).

This does not mean every principle becomes code. Many principles speak to human decisions that no agent takes — "favour buy over build," "prefer open standards." Those keep one audience and stay prose. The discipline is to **decide, per principle, which audiences it has** — and AEAF makes that decision explicit through one new attribute.

8.3 The Enforcement attribute

Every AEAF principle carries an **Enforcement** value. It declares how the principle binds, and therefore which audiences it must serve. TOGAF principles are advisory by default; AEAF principles must say.

ENFORCEMENT VALUE	HOW IT BINDS	AUDIENCE	WHERE IT LIVES
Advisory	Guidance a human weighs as judgment. Honoured by discipline, not by a mechanism.	Human only	The principle catalog; design reviews.
Design-time	Checked at a governed gate before a change ships. A design that violates it does not pass review.	Human (reviewer)	A gate checklist; the architecture review.

ENFORCEMENT VALUE	HOW IT BINDS	AUDIENCE	WHERE IT LIVES
Runtime guardrail	Enforced continuously on agent behaviour while the agent acts, regardless of the agent’s reasoning.	Human and agent	Compiled into a guardrail at an enforcement point.

The three values form a spectrum from pure human judgment to pure machine enforcement. AEAF uses the indigo/amber convention to show where each value sits: advisory principles are amber (human judgment), runtime principles are indigo (machine-enforced), design-time principles sit between — a human judgment exercised at a fixed point.



Figure 8.1 — The Enforcement spectrum. Enforcement runs from amber (advisory — honoured by human judgment) through a design-time gate (a human judgment exercised at a fixed checkpoint) to indigo (a runtime guardrail enforced on the agent regardless of its reasoning). The further right a principle sits, the less it depends on anyone choosing to honour it in the moment, and the more engineering it costs to enforce.

Choosing the Enforcement value

The choice is not free: each step right buys reliability at the cost of engineering effort and the risk of brittle over-enforcement. Use these criteria, in order.

QUESTION	IF YES, LEAN TOWARD
Does an agent take an action that the principle constrains, at runtime , with no human in the moment?	Runtime guardrail
Is the cost of a violation high or irreversible (money moved, data exposed, action taken in the world)?	Runtime guardrail

QUESTION	IF YES, LEAN TOWARD
Can the principle be stated as a checkable condition — a permission, a limit, a class of data, a veto?	Runtime guardrail (else design-time or advisory)
Is the principle satisfied by how a thing is built , and verifiable before it ships?	Design-time
Does the principle guide a human decision that no agent takes, or resist precise expression?	Advisory

Two rules of thumb keep the choice honest. First, **do not promote to runtime what cannot be expressed as a condition** — a principle compiled into a vague or unmeasurable check produces false confidence, which is worse than honest advice. Second, **do not leave at advisory what an agent can violate at speed** — "the agent should be careful with customer data" is decoration if nothing stops the agent from sending it. When a high-consequence principle resists clean expression, the right move is usually to *narrow* it to the part that can be enforced and keep the rest advisory, not to abandon enforcement entirely.

8.4 Compiling a principle into a guardrail

A runtime principle does not enforce itself. It is **compiled** — translated from the human statement into one or more guardrails that a runtime can apply at a named enforcement point. This is the formal expression of "architecture now has a second audience," and it is the single most important mechanical move in this chapter. The meta-model records it as the relationship *Principle* → *compiled into* → *Guardrail* (→ see AEAF Framework §5).

Compilation is a deliberate translation across four questions: **what** the human rule says, **what condition** expresses it precisely, **where** that condition is checked, and **what happens** when it fails. The chain below shows the path; the engineering practice of building guardrails — the patterns, the tooling, the testing — is Book 2 work (→ see AEAF in Practice §9), and the runtime mechanics of how enforcement points actually intercept agent actions are governance, covered in → see AEAF Framework §20.

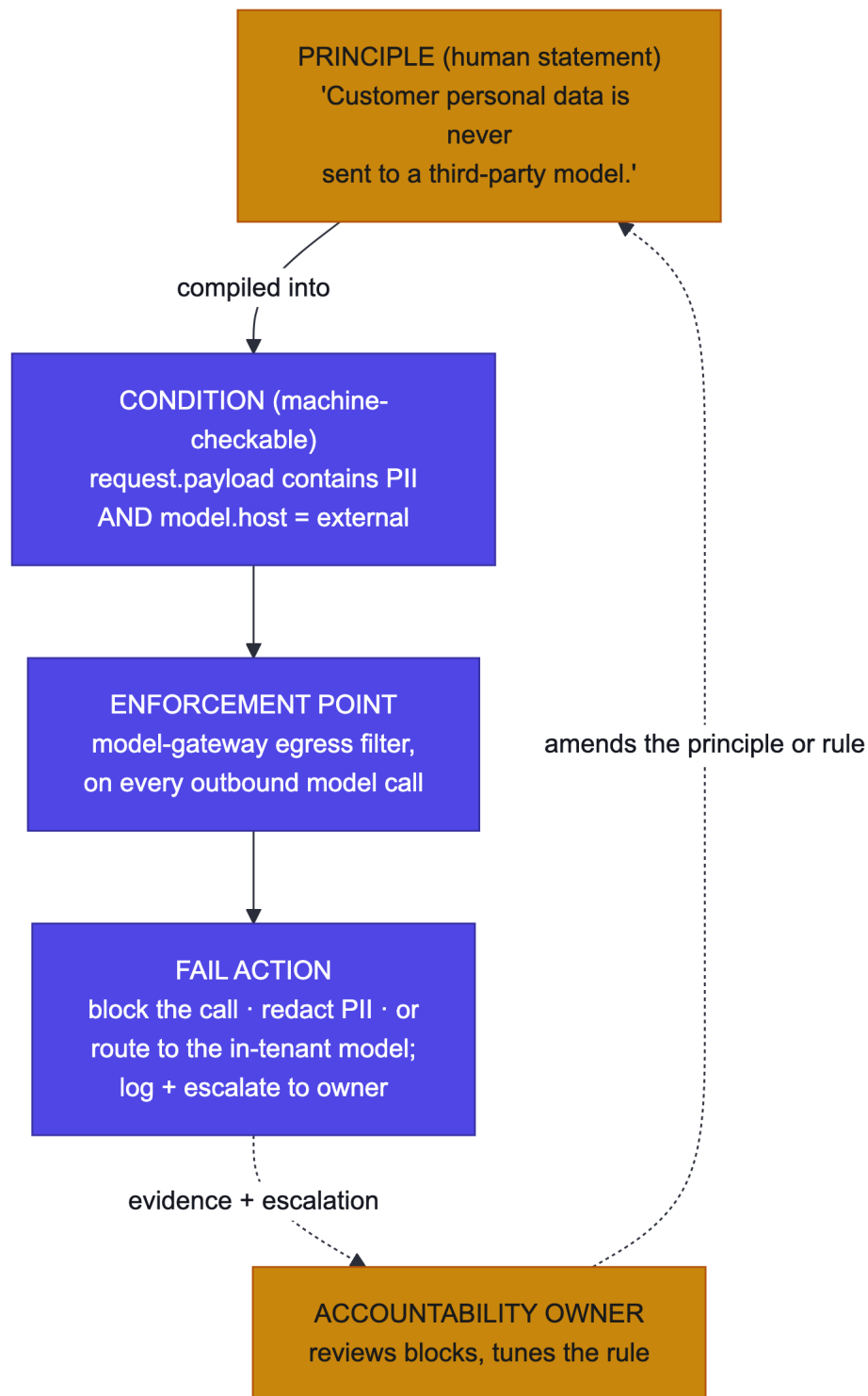


Figure 8.2 — Compiling a principle into a guardrail. The amber human statement at the top is what people read and debate; it is *compiled into* an indigo machine-checkable condition, bound to a named enforcement point with a defined fail action. The loop back to the amber

accountability owner is what keeps the compiled rule honest — a guardrail is not fire-and-forget; a human owns its behaviour and amends the principle when the rule proves too tight or too loose.

Worked example: the PII boundary

Take the principle most often cited for this purpose and run it through the compilation.

COMPILATION STEP	CONTENT
Principle (Name)	Customer data stays inside the trust boundary.
Statement (human)	Customer personal data is never sent to a model or service outside the enterprise’s trust boundary.
Rationale	An agent calling an external model can leak regulated personal data into a third party’s logs or training set; the exposure is irreversible and the enterprise remains the data controller.
Condition (machine)	<code>outbound_call.target ≠ approved_in_boundary_models AND pii_classifier(outbound_call.payload) = true</code>
Enforcement point	The model gateway, on every outbound model call an agent makes.
Fail action	Block the call; attempt PII redaction and retry, or route to an in-boundary model; if neither is possible, escalate to the capability’s accountability owner. Log the event with the offending field classes (not the data itself).
Enforcement value	Runtime guardrail. An agent decides which model to call at runtime, with no human in the moment, and the cost of a violation is high and irreversible.

The example shows what dual-audience means in practice. The human reads the **Statement** and understands the intent; the agent is bound by the **Condition** at the **Enforcement point**, whether or not its reasoning ever considered the principle. The two forms say the same

thing to two audiences. If they ever diverge — if the condition lets through something the statement forbids — that is a defect in the compilation, caught by the eval suite that tests the guardrail and by the owner who reviews its logs.

8.5 The AEAF Principle Catalog

The catalog below is AEAF's reference set of principles for the agentic enterprise. It is not exhaustive and not mandatory in full — like the framework itself it is meant to be **tailored** (→ see AEAF in Practice §2). It is the starting set an organisation adapts: keep what fits, set the Enforcement value to match your risk and your engineering capacity, add your own.

Each principle carries the four classical parts plus **Enforcement**. Where Enforcement is *runtime guardrail*, the principle is a candidate for compilation by the path in §8.4. The principles are grouped into five families.

8.5.1 Foundational principles

These set the ground rules of the human–agent enterprise. They are the principles every other principle assumes.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
F1	Accountability is never delegated	Every capability an agent performs has exactly one named human or legal entity who is answerable for its outcomes; an agent is responsible for execution but never accountable.	An agent has no legal standing and cannot bear consequences. If no human is answerable, the enterprise has created action without responsibility.	One accountability owner recorded per capability before any agent acts in it; recorded in the Trust & Accountability Matrix; an unowned agent capability is a defect, not a state.	Design-time — checked before an agent goes live; the ownership exists is verifiable, the answerability itself is human.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
F2	Humans can always override	For every agent capability there is a defined way for an authorised human to stop, reverse, or take over the work, and it is tested.	Probabilistic actors fail in unforeseen ways; the enterprise must never be in a position where it cannot regain control of an acting agent.	A kill path per agent (disable tool, halt session, roll back); override is exercised in testing, not assumed; out-of-the-loop autonomy still has an override, it just sits behind monitoring.	Runtime guardrail — the stop/override mechanism is a runtime control; its existence is also a design-time gate.
F3	Intent over specification	Agent work is governed by a declared intent — the outcome to achieve and the bounds to stay within — not by an exhaustive specification of steps.	A probabilistic actor cannot be fully specified or exhaustively tested; demanding a complete spec either blocks the work or produces a false sense of control.	Every agent capability has an Intent (outcome, bounds, priority) and an eval suite that measures behaviour against it; "untested" because unspecifiable "is not acceptable — bounds and evals replace the spec.	Advisory — a stance that shapes how the method is run; it is realised through Intent & Assurance (→ see AEA Framework §17).

8.5.2 Safety & trust principles

These bound what an agent is permitted to do and how exposure is contained. They are the principles most likely to compile to runtime guardrails.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
S1	Least-privilege agency	An agent is granted only the tools, data access, and action scope its capability requires, and no more; access defaults to deny.	The blast radius of a misaligned or compromised agent is the set of things it can touch. Minimising that set is the cheapest reduction of consequence available.	Every tool and data binding is explicitly enumerated and justified; read access is separated from write; new permissions require a deliberate grant, not a default.	Runtime guardrail — the tool-permission model enforces scope on every action.
S2	Autonomy proportional to consequence	The autonomy level granted to an agent is set by the consequence and reversibility of its actions, not by the agent's measured capability.	A highly capable agent acting on a high-consequence, irreversible decision is the most dangerous configuration; competence is not a licence for unsupervised action.	High-consequence or irreversible actions require a lower autonomy level (in-the-loop or on-the-loop); the autonomy level is recorded per allocation and reviewed when consequence changes.	Design-time — the autonomy level is set and reviewed at a gate; the oversight mode it implies is a runtime control.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
S3	Defense in depth	No single control is trusted to bound an agent; input, model, reasoning, output, and operational controls are layered so a failure of one is caught by the next. ¹	Every individual control fails sometimes — a model is jailbroken, a filter misses an input, a guardrail has a gap. Layered controls turn a single failure into a contained one.	Controls are designed across the five layers, not concentrated at one; an agent capability with a single point of control is under-protected; defense depth is part of the assurance case.	Design-time — the layering is an architecture decision verified at a gate; individual layers are runtime guardrails.
S4	Data stays inside the trust boundary	Customer and regulated personal data is never sent to a model or service outside the enterprise’s defined trust boundary.	An agent calling an external model can irreversibly leak regulated data into a third party; the enterprise remains the data controller regardless of which agent made the call.	PII classification on outbound model calls; an approved in-boundary model list; egress filtering at the model gateway; blocks logged and reviewed. (Worked compilation in §8.4.)	Runtime guardrail — enforced on every outbound model call.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
S5	An agent's reach is bounded	Every agent operates within explicit limits on spend, action volume, and rate, beyond which it cannot act without escalation.	A tireless actor at machine speed turns a small error into a large one in seconds; loops, runaway costs, and mass-action mistakes are bounded only by hard limits.	Per-agent and per-session budgets (spend, tool calls, write operations, rate); exceeding a budget triggers escalation, not silent continuation; budgets are sized to the capability and reviewed.	Runtime guardrail — budgets and rate limits are enforced at the operations layer.

8.5.3 Assurance principles

These replace "build it correctly, test it once" with continuous evidence that an agent behaves within bounds over time.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
A1	Evaluate continuously, not once	Agent behaviour is measured against its intent on a maintained eval suite, on an ongoing cadence, not verified once at acceptance and assumed thereafter.	A probabilistic actor’s behaviour drifts as models, prompts, data, and the world change; a one-time acceptance test is stale the day after it passes.	Every agent capability has an eval suite (quality, safety, alignment, cost) with thresholds and a cadence; failing an eval is an incident; evals shape, not just audit, the assurance case (→ see AEAF Framework §17).	Design-time + ongoing — the eval suite is gated at deployment; its continuous running is a runtime assurance control.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
A2	Fail safe, fail closed	When an agent is uncertain, hits a bound, or a control fails, it stops and hands control to a human rather than guessing and proceeding.	The default behaviour of a tireless actor under uncertainty must be to not act; a probabilistic actor that proceeds on low confidence converts uncertainty into consequence.	Low confidence, guardrail veto, and control failure all route to a defined escalation path; the unsafe default (proceed anyway) is never the fallback; escalation is a designed path, not an error.	Runtime guardrail — the fail-closed behaviour is enforced at the decision and output layers.
A3	Observable by default	Every consequential agent action is logged with enough context — inputs, decision, tool calls, outcome — to reconstruct what happened and why, before the agent ships.	You cannot govern, assure, or investigate what you cannot see; an agent acting at machine speed without an audit trail is ungovernable by construction.	Action-level audit logging on by default; reasoning and tool calls traceable; logs retained for the assurance and incident needs of the capability; observability is built in, not added after an incident.	Design-time — observability is a verified property of the build; the logging itself runs at runtime.

8.5.4 Workforce principles

These govern the human-agent relationship — oversight, the people whose work changes, and the people affected by agent action.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
W1	Oversight proportional to consequence	The oversight mode for a capability — in-the-loop, on-the-loop, or out-of-the-loop — is set by the consequence, reversibility, and confidence of the work, and is reviewed as these change.	Uniform oversight is wrong in both directions: it wastes human attention on trivial actions and starves it from consequential ones. Oversight is a scarce resource to be allocated, not a constant.	Each capability allocation records its oversight mode and the trigger that would change it; high-consequence work keeps a human in or on the loop; the mode is reviewed when confidence or consequence shifts.	Design-time — the mode is set and reviewed at a gate; it pairs with S2 (autonomy).

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
W2	Reskill alongside automation	When a capability moves across the automation frontier from human to agent, the architecture plans the human transition — redeployment, reskilling, or role change — as part of the same decision.	Moving the frontier is the era's central transformation lever, and its human cost is foreseeable; an enterprise that automates without planning the transition trades a known short-term gain for trust and capability it will need later.	A frontier move carries a human-transition plan, not only a technical one; the people whose work changes are identified and planned for in the migration (→ see AEAF Framework §16); silent displacement is a governance failure.	Advisory — a planning obligation on the people who move the frontier; not machine-enforceable.
W3	Transparency to affected people	A person interacting with, or materially affected by, an agent's action is told that an agent was involved and given a path to a human.	Trust depends on people knowing when a non-human actor is acting on them; concealment is both an ethical failure and, increasingly, a regulatory one. ²	Agent involvement is disclosed at the point of interaction; an escalation path to a human is available and findable; disclosure is designed into the interaction, not buried in terms.	Runtime guardrail where the interaction is machine-mediated (the disclosure is emitted by the system); advisory where the obligation falls on human design.

8.5.5 Architecture principles

These govern how AEAF artifacts themselves are made — the principles the framework holds itself to.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
R1	Normative artifacts are dual-audience	Any architecture artifact that constrains agent behaviour carries both a human statement and a machine-enforceable form, kept in correspondence.	An artifact an agent must obey at runtime but which exists only as prose cannot bind the agent; the second audience needs a form it can act on (§8.2).	Principles meant to bind agents declare an Enforcement value; runtime ones compile to guardrails; the human and machine forms are tested for correspondence and amended together.	Design-time — verified when the artifact is authored or changed.
R2	Tailorable by design	AEAF and its principle catalog are adopted as a coherent subset and grown into; no organisation is required to take all of it at once.	A framework that must be adopted whole is adopted by no one; the agentic enterprise exists at every scale, from startup to regulated incumbent, and each needs a different subset.	This catalog is a starting set, not a mandate; an organisation keeps what fits and sets Enforcement to match its risk and capacity; what depends on what is mapped so a subset stays coherent (→ see AEAF in Practice §2).	Advisory — a stance on how the framework is used.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
R3	Evidence over hype	AEAF claims are falsifiable, its trade-offs named, and its failure modes stated; the framework earns trust the way it asks agents to earn it.	A framework for governing probabilistic actors cannot credibly demand evidence of them while asserting itself on confidence; the discipline must be visible in the framework’s own voice.	Claims of fact are sourced; trade-offs of each principle are stated, not hidden; named failure modes accompany recommendations; intensifiers and unfalsifiable claims are removed (the style guide enforces this on the prose).	Advisory — an editorial and intellectual discipline.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
R4	Technology & vendor independence	The architecture commits to capabilities and option-classes, never to a specific vendor or product; any vendor or product named is illustrative only, and vendor lock-in is a recorded, justified trade-off, never a default.	A framework that binds itself to a named product inherits that product’s lifespan, pricing, and roadmap; the agentic technology landscape moves too fast for a standard to be a procurement list and survive.	Architecture decisions name the option-class and the selection criteria, not the winner; where a vendor is named it is marked as illustration; any deliberate lock-in is documented with its rationale and exit cost, reviewed when the market moves (→ see AEAF Framework §14).	Advisory + design-time — an editorial stance on how the framework is written, checked at a gate when a design names a specific product as a commitment.

#	NAME	STATEMENT	RATIONALE	KEY IMPLICATIONS	ENFORCEMENT
R5	Principle-led, protocol-pluggable	New connecting technologies and protocols are adopted as capabilities an agent reaches through guardrailed actions, governed by autonomy level and accountability owner — never as framework dependencies; AEAF commits to principles and option-classes, not to any single protocol (e.g. MCP today).	A protocol named in the framework becomes load-bearing the moment a reader treats it as required; the connecting layer turns over faster than the principles do, so binding to today's protocol dates the standard and couples it to one ecosystem's fortunes.	A new protocol enters as an option-class assessed on the Technology Radar (→ see AEAF Framework §14), not as a clause; an agent reaches it through a bounded, guardrailed action under a set autonomy level and a named accountability owner; the portability principle is the commitment, the specific protocol is the current realisation.	Advisory + design-time — a stance on how connecting technologies enter the framework, checked at a gate when a design would make one protocol a dependency.

Reading the catalog

Eighteen principles, five families. The distribution of Enforcement values is itself the chapter's argument: the **safety & trust** family compiles most often to runtime guardrails, because that is where agents take consequential action with no human in the moment; the **architecture** and **workforce** families stay mostly advisory, because they govern human

decisions about how the framework is used and how the frontier is moved. **Foundational** and **assurance** principles cluster at design-time, with runtime components where an agent's live behaviour is what must be bound.

FAMILY	PRINCIPLES	DOMINANT ENFORCEMENT
Foundational	F1–F3	Design-time, with runtime override (F2)
Safety & trust	S1–S5	Runtime guardrail
Assurance	A1–A3	Design-time + ongoing / runtime
Workforce	W1–W3	Mixed (design-time and advisory)
Architecture	R1–R5	Advisory / design-time

8.6 How principles relate to the rest of AEAF

Principles are not free-standing. They are the upstream source of two of the framework's central mechanisms, and they are downstream of the enterprise's drivers and intent. Three relationships matter most.

Principles → guardrails → governance. A runtime principle compiles into a guardrail (§8.4); the guardrail is then enforced, monitored, tuned, and audited as part of continuous and runtime governance (→ see AEAF Framework §20). The principle is the belief; the guardrail is the mechanism; governance is the loop that keeps the mechanism honest and catches it when it drifts. The engineering practice of building and operating guardrails is Book 2 (→ see AEAF in Practice §9 and §14); this chapter ends at the compilation boundary deliberately.

Principles ↔ the Intent & Assurance centre. The AEAF Method puts Intent & Assurance where TOGAF put Requirements (→ see AEAF Framework §9 and §17). Principles bound that centre on both sides: an Intent is declared *within* the bounds the principles set (F3, S2, S4, S5), and the eval suites that assure the Intent test for the very behaviours the safety and assurance principles forbid or require (S3, A1, A2, A3). A principle without a corresponding guardrail or eval is advice; a guardrail or eval without a principle behind it is a rule no one can explain. The chains hold the two together.

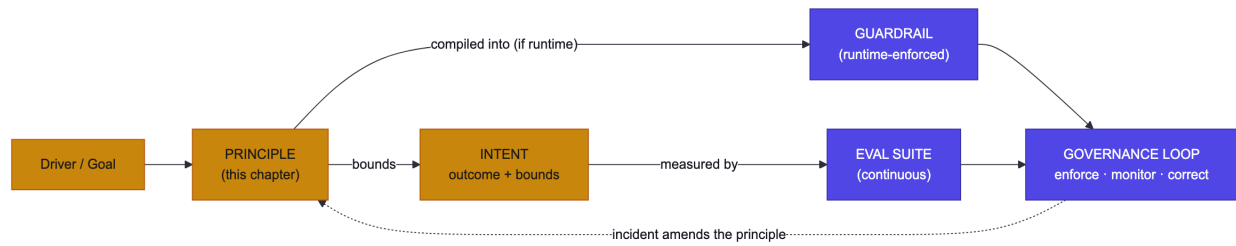


Figure 8.3 — Where principles sit in the AEAF traceability spine. Principles (amber) descend from drivers and goals, set the bounds an Intent is declared within, and — when runtime-enforced — compile into guardrails (indigo). Guardrails and the eval suites that measure Intent both feed the governance loop, which closes back to the principle when an incident shows the belief, the rule, or the bound was wrong. This is the chain the meta-model requires to stay navigable (→ see AEAF Framework §5).

Principles → the catalog as an artifact. The catalog in §8.5 is a deliverable in its own right, maintained over time and recorded alongside the Guardrail/Policy Catalog and the Trust & Accountability Matrix in the AEAF artifact set (→ see AEAF Framework §21; templated in → see AEAF in Practice §15). A principle that binds agents is not a sentence in a slide; it is a governed object with an Enforcement value, a compilation target, and an owner — which is what it means to take seriously the fact that architecture now has a second audience.

1. The five-layer frame — input, model, reasoning, output, operations — follows established defence-in-depth practice for LLM-based systems. See OWASP Top 10 for LLM Applications (current version) and NIST AI Risk Management Framework (AI RMF 1.0, 2023). [↗](#)
2. Disclosure obligations for AI systems that interact with people are codified in the EU AI Act, Article 50 (transparency obligations), Regulation (EU) 2024/1689. [↗](#)

Part II • The Method

Chapter 9 — The AEAF Method: overview & the cycle

In brief. Enterprise architecture needs a method — a repeatable way to move from where the organization is to where it intends to be, under governance. The AEAF Method is that method for the Agentic Enterprise: the framework's answer to TOGAF's Architecture Development Method (ADM). It keeps the proven shape — a governed, iterative cycle that runs from a Preliminary phase through Vision to implementation and change — because that shape works. It makes three changes that the agentic enterprise forces. First, the centre of the method is **Intent & Assurance Management**, not Requirements Management, because a probabilistic actor is bounded and evaluated, not specified and verified. Second, a new phase — **Intelligence & Agency (Phase 6)** — is added to architect the population of models, agents, knowledge, guardrails, and evals; it is the heart of the method. Third, governance is no longer only a gate at the end of a cycle: a **runtime assurance loop** runs continuously while agents operate. The method is therefore a **dual loop** — a deliberate design-time cycle joined to a fast runtime cycle at governance. This chapter gives the overview and the cycle; the per-phase detail is in Chapters 10–17, the centre in Chapter 17, and the governance loop in Chapter 20.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where TOGAF uses the ADM, AEAF uses the AEAF Method; the phase-to-ADM mapping in this chapter is offered as a migration aid for existing TOGAF practices, not as a claim of lineage.

9.1 Why a method, and why this one

A framework is a set of concepts; a method is what you do with them. The preceding chapters established the concepts the Agentic Enterprise needs — the reworked Actor model (→ see AEAF Framework §7), the five BDAT+I domains (→ see AEAF Framework §6),

and dual-audience principles (→ see AEAF Framework §8). This part of the book turns those concepts into work: a phased, cyclic procedure that an architecture team follows to design, allocate, govern, and assure the blended workforce.

The AEAF Method is that procedure. It is deliberately built as the agentic counterpart to TOGAF's ADM rather than as a clean-sheet invention, for one reason: the ADM's core discipline is sound and worth keeping. Architecture still moves from a baseline to a target. It still does so in a governed, iterative cycle. It still draws every phase toward a stable centre. An organization that runs the ADM today should recognize the AEAF Method as a familiar machine with three deliberate modifications — not as something it must learn from nothing.

Those three modifications are not cosmetic. Each answers one of the four foundational breaks set out in the founding thesis (→ see AEAF Framework §3), and together they are what make this a method for actors that decide and act rather than for systems that merely run.

CHANGE	WHAT IT REPLACES	THE BREAK IT ANSWERS
Centre becomes Intent & Assurance Management	Requirements Management at the centre	A3 — probabilistic actors cannot be fully specified or exhaustively tested.
New phase 6 • Intelligence & Agency	(no analogue in the ADM)	A1 — agents are a new kind of actor needing their own architecture domain.
Runtime assurance loop added to governance	Gate-only, episodic governance	A4 — agents act and drift at machine speed, between governance meetings.

The remaining changes — performer-type on roles, the cognitive substrate in data, design for non-determinism in applications — fall out of these three as each classical phase is reworked. The chapters in Part III take each phase in turn. This chapter stays at the level of the whole.

9.2 The cycle

The AEAF Method is a **dual loop**. The outer loop is the deliberate, governed design-time cycle: a Preliminary phase that stands up the capability, then ten phases that run from architecture vision through to change. The inner loop is the runtime assurance loop: a fast,

continuous cycle that operates while agents are live, enforcing guardrails and monitoring behaviour at machine speed. The two loops meet at governance — the design-time gate hands an approved agent to the runtime loop, and material change in the runtime loop triggers a fresh design-time cycle.

Figure 9.1 shows the whole method. The outer ring will be recognizable to any ADM practitioner: each phase carries its ADM counterpart in parentheses so the mapping is visible at a glance. The inner loop and the new Phase 6 are AEAF's additions. Intent & Assurance Management sits at the centre and feeds every phase and both loops, exactly as Requirements Management touched every ADM phase.

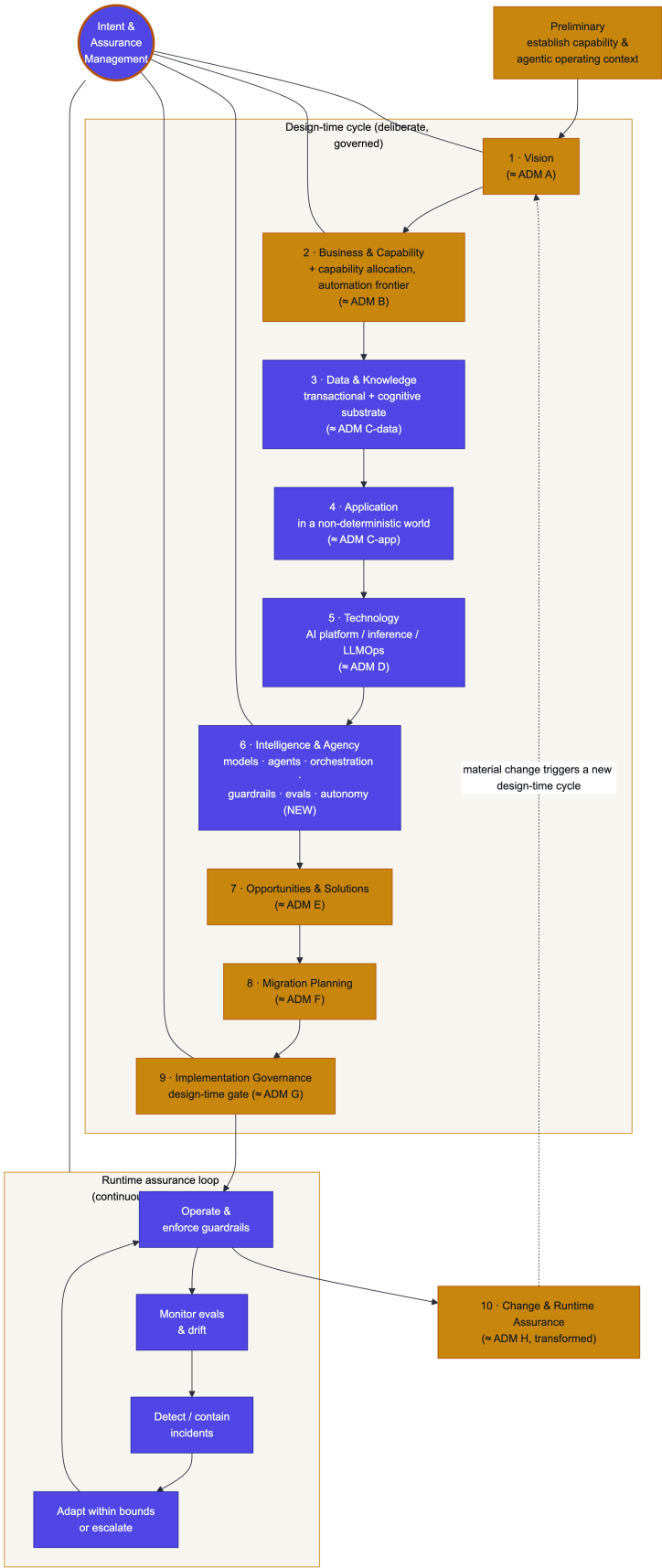


Figure 9.1 — The AEAF Method. The outer ring is the deliberate design-time cycle, with each phase mapped to its ADM counterpart and one new phase (6 · Intelligence & Agency) added. The inner loop is the runtime assurance loop, which runs continuously while agents operate. Intent & Assurance Management sits at the centre, feeding every phase and both loops. Indigo nodes are where the work is predominantly machine-facing (the technical and intelligence phases, and the whole runtime loop); amber nodes are where human judgment leads (vision, allocation, migration decisions, the design-time gate, and change). The centre carries both colours because intent is set by humans and assurance is measured against machine behaviour.

Three properties of the cycle deserve emphasis before the phases are walked.

- **It is genuinely a loop, not a waterfall.** The phases are drawn in sequence for legibility, but the method iterates and runs concurrently across capabilities — exactly as the ADM does. A copilot for one team can be in its runtime loop while a more consequential agent for another is still in design.
- **The two loops run on different clocks.** The design-time cycle turns in weeks or months, on the cadence of architecture work. The runtime loop turns in seconds, on the cadence of agent decisions. Treating them as one loop is the modelling error that gate-only governance makes.
- **Material change closes the outer loop.** The dotted return arrow from Phase 10 back to Vision is the trigger that makes the method continuous: a new capability, a widened autonomy level, or a failed assurance case sends the affected slice back through design-time governance.

9.3 The centre: Intent & Assurance Management

In the ADM, Requirements Management sits at the centre and every phase reads and writes requirements. The AEAF Method keeps a centre — every phase still draws toward one stable point — but generalizes what that centre holds. For deterministic work, a requirement is still a requirement, specified up front and verified by testing. For probabilistic work — anything an agent or model does — a fully specified requirement is neither achievable nor sufficient, because the same input can yield different actions and the behaviour space cannot be enumerated.

The centre therefore manages three linked things, not one. This chapter previews them; the full treatment, including the assurance case and how it is built and refreshed, is in Chapter 17 (→ see AEAF Framework §17).

ELEMENT OF THE CENTRE	WHAT IT HOLDS	EXAMPLE
Intent	The declared desired outcome, with bounds, priority, and an acceptance signal — what good looks like, not the steps to get there.	"Resolve refund requests within policy; never exceed €X without human approval; aim for ≥ 95% customer-acceptable resolutions."
Guardrails	The constraints the work must not violate, derived from principles, standards, and regulation; some advisory, some enforced at runtime.	A spend limit, a PII filter, a tool permission, an action veto on the refund agent.
Assurance	The eval suites, thresholds, and evidence — the assurance case — that show intent is being met within bounds, refreshed continuously rather than signed off once.	An eval suite of refund scenarios run on every model change, plus live guardrail logs and oversight records.

Every phase contributes to and draws from this centre. Vision declares the intent and the risk envelope; Business & Capability allocates capabilities against it; Intelligence & Agency designs the agents and their evals to meet it; Implementation Governance approves the assurance thresholds; and the runtime loop produces the assurance evidence that keeps the case alive. Where TOGAF's centre asked "have we met the requirement?", AEA's centre asks the continuing question "is this actor still behaving within intent, on the evidence we have now?"

9.4 The phases at a glance — the migration bridge

The AEA Method has a Preliminary phase and ten numbered phases. Eight of the numbered phases map directly to an ADM phase and are reworked rather than replaced; one (Phase 6) is wholly new; and one (Phase 10) is the ADM's change-management phase transformed into the owner of the runtime loop. Table 9.1 walks all of them at overview level. It is the **migration bridge**: an existing TOGAF practice can read down the "ADM letter" column to find the phase it already knows, then read across to see what is new. The per-phase chapters listed in the final column carry the detail.

Table 9.1 — The AEA Method phases, mapped to the ADM (the migration bridge).

#	PHASE	ADM LETTER	PURPOSE (ONE LINE)	WHAT'S NEW IN THE AGENTIC ENTERPRISE	DETAIL IN
P	Preliminary	Preliminary	Stand up the architecture capability; establish principles, governance, and repository; characterize the agentic operating context and risk appetite.	Add agentic principles (including runtime-enforced ones); define the autonomy-level scheme and accountability model up front.	Ch.4–8
1	Vision	A	Set drivers, stakeholders, scope, and target value; produce the architecture vision and secure approval.	State the human-agent ambition and the consequence/risk envelope that bounds autonomy.	Ch.10
2	Business & Capability	B	Model capabilities, value streams, processes, and organization.	Capability allocation (human / agent / hybrid) and the automation-frontier map become primary outputs; roles gain a performer-type.	Ch.11

#	PHASE	ADM LETTER	PURPOSE (ONE LINE)	WHAT'S NEW IN THE AGENTIC ENTERPRISE	DETAIL IN
3	Data & Knowledge	C (data)	Design conceptual, logical, and physical data; governance and MDM.	Add the cognitive substrate — knowledge corpora, embeddings, memory, graphs — with their classification and freshness.	Ch.12
4	Application	C (app)	Design the application portfolio, integration, and services.	Design for non-determinism : bounded outputs, fallback paths, human handoff, idempotency, and observability of probabilistic components.	Ch.13
5	Technology	D	Design infrastructure, platforms, and security.	The AI platform : inference, model gateways, vector stores, orchestration / agent runtime, and the LLMOps substrate.	Ch.14

#	PHASE	ADM LETTER	PURPOSE (ONE LINE)	WHAT'S NEW IN THE AGENTIC ENTERPRISE	DETAIL IN
6	Intelligence & Agency	— (new)	Architect the population of models, agents, orchestration, knowledge use, guardrails, evals, and autonomy levels — the +I domain.	The whole phase is new. This is where agents are designed as actors, not as features.	Ch.15
7	Opportunities & Solutions	E	Define work packages, transition architectures, and value increments.	Sequence by trust earned : pilot under tight oversight, widen autonomy as assurance accrues.	Ch.16
8	Migration Planning	F	Sequence work; manage dependencies, risk, and the business case.	Plan frontier moves (re-allocations) and the human transition (reskilling, role change) explicitly.	Ch.16

#	PHASE	ADM LETTER	PURPOSE (ONE LINE)	WHAT'S NEW IN THE AGENTIC ENTERPRISE	DETAIL IN
9	Implementati on Governance	G	Run compliance reviews and architecture contracts; operate the design-time gate.	The gate also approves guardrails, autonomy levels, eval thresholds, oversight mode, and the handoff to the runtime loop.	Ch.20
10	Change & Runtime Assurance	H (transformed)	Monitor, assess impact, evolve; trigger new cycles.	Owns the continuous runtime loop: enforcement, drift and eval monitoring, incident response, bounded adaptation, and escalation.	Ch.20

Two reading notes on the bridge. First, the mapping is fair to TOGAF: each ADM phase is named accurately for what it does before AEA states what it changes, and Phase 6 is shown honestly as having no ADM analogue rather than being forced onto one. Second, the bridge is carried forward and made operational in the adoption guide (→ see AEA in Practice §2) and in the migration chapter (→ see AEA Framework §23), so a TOGAF shop can adopt AEA phase by phase.

9.5 Phase 6 — Intelligence & Agency, the heart of the method

One phase has no ADM counterpart, and it is the reason the method exists in this form.

Phase 6 · Intelligence & Agency is where the architecture designs the population of the +I domain: the models in use, the agents built on them, how those agents are orchestrated, the knowledge corpora they retrieve from, the guardrails that bound them, the eval suites that measure them, and the autonomy levels they are granted.

In a classical EA practice these concerns have no home. Business, Data, Application, and Technology each has a natural owner, but none of them naturally owns "the set of agents that act in the enterprise, what they may do, and how we know they are behaving." Folding agents into the Application domain — treating an agent as just another box in the application portfolio — is the central modelling error the founding thesis identifies (→ see AEAF Framework §3). Phase 6 corrects it by giving agents, as actors, their own design phase.

Phase 6 is flagged here as the heart of the method, but its detail is deferred. The full treatment — the Agent Catalog, Model Portfolio, Knowledge-Corpus Catalog, Guardrail/Policy Catalog, Eval-Suite Specification, and Autonomy-Level Classification, and how they are produced — is Chapter 15 (→ see AEAF Framework §15). What matters at overview level is its position in the cycle: it follows the Technology phase (the AI platform must exist for agents to run on) and precedes Opportunities & Solutions (so that sequencing can be driven by the autonomy and assurance the agents are designed for).

9.6 The dual governance loop

Governance is the method's defining change and the answer to broken assumption A4. In TOGAF, governance is concentrated in Phase G as a gate: architecture contracts, compliance review, sign-off. AEAF keeps that gate and adds a second mechanism that the ADM has no equivalent for — a governance loop that runs while the system runs.

- **Design-time governance (the gate).** As in ADM Phase G — architecture contracts, compliance review, sign-off. AEAF extends the gate so that it also approves the agent's **guardrails, autonomy level, eval thresholds, oversight mode, and accountability owner** before the agent is allowed to operate. Nothing crosses into runtime without passing this gate.
- **Runtime governance (the loop).** Once operating, an agent is governed *while it runs*. Guardrails are enforced at each decision point; eval suites and drift detectors monitor

behaviour continuously; incidents are detected and contained; and the agent adapts within its bounds or **escalates** to a human. Material change — a new capability, a widened autonomy level, or a failed assurance case — triggers a new design-time cycle.

This overview is enough to read the cycle. The full design of both halves — what the gate checks, how enforcement points are placed, how drift is detected, and how incidents and evidence are handled — is Chapter 20 (→ see AEA Framework §20).

The lifecycle of a single agent under dual governance is shown in Figure 9.2. An agent is proposed, gated at design time, handed to runtime, and kept within bounds while operating; a breach contains it; a material breach routes it back into a new design-time cycle; and at end of life it is retired.

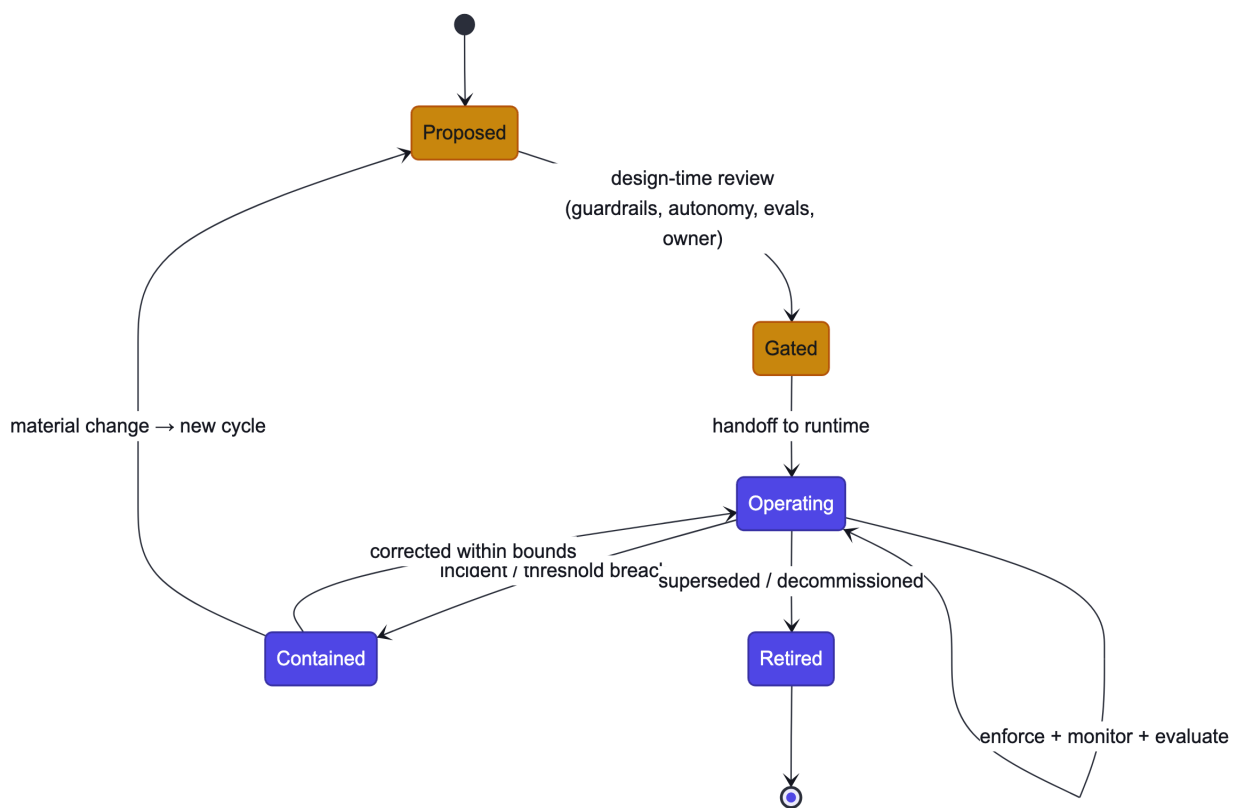


Figure 9.2 — Lifecycle of an agent under dual governance. Design-time gating (amber: human judgment proposes and approves) precedes operation; the runtime loop (indigo: machine-speed enforcement, monitoring, and containment) keeps the agent within bounds; a breach contains the agent, and a material breach routes it back into a new design-time cycle. The path from *Operating* to *Contained* and back is the runtime loop's normal correction; the path from *Contained* to *Proposed* is the dotted return arrow of Figure 9.1 seen at the level of one agent.

9.7 Tailoring the method

The AEAF Method, like the ADM, is meant to be selected, tailored, and partially adopted — not run whole on every engagement. The framework's design tenets (→ see AEAF Framework §3) make tailorability a first-class property, and the full tailoring guidance is in the adoption guide (→ see AEAF in Practice §2). Four rules govern tailoring at the level of the method.

RULE	GUIDANCE
Scope to the work.	A single copilot deployment needs Phases 2, 6, and 9–10 plus the centre; a full enterprise transformation uses all of them. Use the smallest coherent subset that still answers who performs the capability, how it is bounded, and how it is assured.
Phase 6 and the runtime loop are mandatory where agents act.	If any actor in scope is an agent, the Intelligence & Agency phase and the runtime assurance loop are non-negotiable — they are where autonomy, guardrails, and assurance live. Dropping them is not tailoring; it is leaving agents ungoverned.
Match governance weight to consequence.	Governance weight scales with the consequence of the agent's actions. A low-consequence, reversible, in-the-loop agent needs light gating; a high-consequence or out-of-the-loop agent needs the full design-time gate and a strong runtime loop.
Iterate and run concurrently.	The cycle iterates and runs concurrently across capabilities, exactly as the ADM does. The design-time cycle turns at the cadence of architecture work; the runtime loop never stops once anything is operating.

The first rule keeps the method proportionate; the second keeps it safe; the third keeps governance from being either ceremony or negligence; and the fourth keeps it continuous. Together they let an organization adopt a coherent slice of the method and grow into the rest as its blended workforce grows — which is the subject of the maturity model (→ see AEAF Framework §22) and the adoption archetypes (→ see AEAF in Practice §3).

This chapter gave the method at the level of the whole cycle. The phases are detailed in Chapters 10–16, the centre in Chapter 17, and the dual governance loop in Chapter 20. The phase-to-ADM bridge in Table 9.1 is carried forward into the migration chapter (→ see AEAF Framework §23) and operationalized in the adoption guide (→ see AEAF in Practice §2).

Chapter 10 — Vision

In brief. The Vision phase is where an architecture cycle begins: it names the drivers, identifies the stakeholders, fixes the scope, states the target value, sets out an architecture vision at a level a sponsor can approve, and secures that approval before detailed work starts. AEAF keeps all of this — it is the right way to start. The agentic enterprise adds two outputs that did not exist when the only actors were human, and both must be stated here because they frame every later phase. The first is the **human-agent ambition**: a plain statement of *where the enterprise wants agents doing work, and to what end*. The second is the **risk/autonomy appetite**: the **consequence/risk envelope** the enterprise is willing to operate inside, and how much independent action it is willing to grant. Stated late, these are post-rationalisation; stated here, they bound Business & Capability allocation (→ see AEAF Framework §11), Intelligence & Agency design (→ see AEAF Framework §15), and governance (→ see AEAF Framework §20). This chapter also makes explicit a problem with no classical analogue: in the agentic enterprise some stakeholders are not only consulted about the architecture — they are *affected by its autonomy*. The workforce whose work is reallocated, the customer a bounded agent acts on, the regulator who will ask who was answerable — these are stakeholders in a stronger sense than TOGAF's term carries.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

10.1 What the Vision phase is for

The Vision phase answers a single question before any domain work begins: *what are we doing this cycle, for whom, why, and within what bounds — and has the sponsor agreed?* It is AEAF Method Phase 1, the counterpart of ADM Phase A. Where TOGAF's Phase A produces a Statement of Architecture Work and an Architecture Vision, AEAF's Vision phase produces the same backbone and two new framing outputs.

The phase exists to prevent two opposite failures. Starting detailed architecture without an agreed vision wastes the cycle on work no one sponsored. Starting with too much detail freezes choices that the later phases should make on evidence. Vision sits deliberately at low resolution: it commits to *direction, scope, value, and bounds*, and defers *how* to the phases built to decide it.

The classical backbone of the phase is unchanged, and AEAF retains each element exactly:

ELEMENT	WHAT IT FIXES	WHY IT MUST BE SETTLED FIRST
Drivers	The business or external forces making this cycle worth doing now.	Without a driver the cycle has no priority and no test of success.
Stakeholders	Who has a stake, what they care about, and their power and interest.	Concerns not gathered here surface late as rework or as objection at the gate.
Scope	The capabilities, organisation, and landscape in and out of this cycle.	Unbounded scope makes the later phases unaffordable and the gate unwinnable.
Target value	The outcome the cycle is meant to deliver, in terms the sponsor recognises.	The value statement is what the final gate measures the work against.
Architecture vision	A sponsor-level sketch of the target — enough to approve, not to build.	A vision a sponsor cannot grasp cannot be approved; one too detailed pre-empts later phases.
Secured approval	The sponsor's authorisation to proceed, recorded.	The cycle has no mandate until someone accountable has said yes.

10.2 What is new in the agentic enterprise

In an enterprise of human actors, Vision needed to state intent, scope, and value — the *consequence* of the work was assumed to flow through people who exercise judgment, and risk was managed by ordinary project discipline. The agentic enterprise breaks that assumption (→ see AEAF Framework §3). When an **agent actor** — a software system that

pursues a goal through tools within guardrails, at a defined autonomy level — performs work, two things must be decided before any domain phase, because every later phase reads them as a constraint.

First, the human-agent ambition. This is a plain statement of where the enterprise *wants* agents doing work and to what end — the strategic shape of the blended workforce this cycle is reaching for. It is not a list of features and not a technology choice. It answers: which kinds of work should agents take on, which work stays human by intent, and what the enterprise is trying to gain (speed, scale, cost, consistency, coverage, or freeing human judgment for higher-value work). Without it, capability allocation in Phase 2 has no north star and degenerates into "automate whatever is easiest," which is how an enterprise drifts into autonomy it never chose.

Second, the risk/autonomy appetite. This pairs two declarations the enterprise must own at the top:

- The **consequence/risk envelope** — the kinds and sizes of consequence the enterprise is willing to let agents create. Money moved without a human signature, customer-facing actions taken unreviewed, irreversible changes, regulated decisions: the envelope says which of these are on the table this cycle and which are off it.
- The **autonomy appetite** — how much independent action the enterprise is willing to grant in principle, expressed against the AEAF Autonomy-Level Classification (→ see AEAF Framework §15) and the three oversight modes (in-the-loop, on-the-loop, out-of-the-loop). Appetite is a ceiling stated up front, not a level assigned to any one capability; the per-capability level is set later, but it can never exceed the appetite agreed here.

These two outputs are not refinements of the classical vision — they are the agentic-enterprise additions that make the rest of the method governable. The reason they belong in Vision and nowhere later is sequence: they are constraints, and a constraint discovered after the work it should have bounded is not a constraint but a regret.

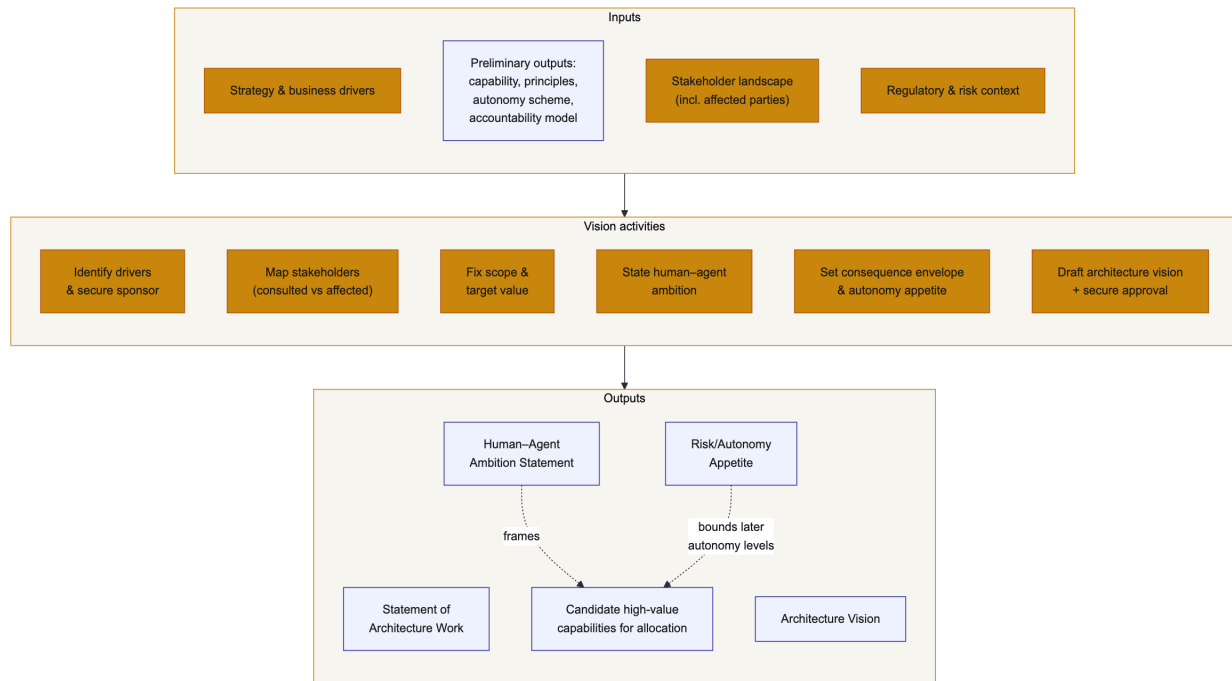


Figure 10.1 — The Vision phase: inputs, activities, outputs. Every node is amber: Vision is performed by human actors exercising judgment, and no agent acts in this phase — agents are its *subject*, not its performer. The two new outputs (Human-Agent Ambition, Risk/Autonomy Appetite) are produced here precisely so they can frame and bound the candidate capabilities that Phase 2 will allocate.

10.3 Stakeholder management when stakeholders are affected by autonomy

Classical Vision treats stakeholders as parties whose **concerns** the architecture must address — they hold a viewpoint, and the architecture answers it. That framing still applies, but it is no longer sufficient. The agentic enterprise creates a second class of stakeholder: parties who are not merely consulted about the architecture but **acted upon by its autonomy**. The distinction matters because the two classes are managed differently, and missing the second is how a cycle produces a technically sound architecture that the workforce resists, a customer distrusts, or a regulator stops.

STAKEHOLDER	STAKE IN CLASSICAL EA	ADDED STAKE IN THE AGENTIC ENTERPRISE
Sponsor / executive	Funds and authorises the cycle; owns the business outcome.	Now also owns the autonomy appetite and the consequence envelope — the bounds are an executive decision, not a technical one.
Workforce	Affected by process change; consulted on requirements.	Affected by capability reallocation — work moves to agents, roles change, some judgment is displaced. A party whose work the architecture redistributes.
Customers	Recipients of the service; concerns gathered indirectly.	Acted upon directly by agent actors (a bounded agent that books, refunds, advises). Their stake is in being treated correctly and being able to reach a human.
Accountable owners	Often implicit in the org structure.	Become explicit and named : exactly one human or role is answerable per capability, including for agent actions, and they must agree to that accountability.
Regulators	A compliance constraint to be satisfied.	A stakeholder who will ask, after the fact, who was answerable for this agent's action — and whose answer the architecture must be able to give.

The practical move in Vision is to map stakeholders on the familiar power/interest grid, but to add a third encoding alongside the two classical axes: whether the party is **consulted** (their concerns shape the architecture) or **affected** (the architecture's autonomy acts on them, or moves their work). Affected stakeholders cannot be managed by communication alone; their stake is changed by the cycle's outcome, so they belong in the appetite conversation, not only in the requirements one.

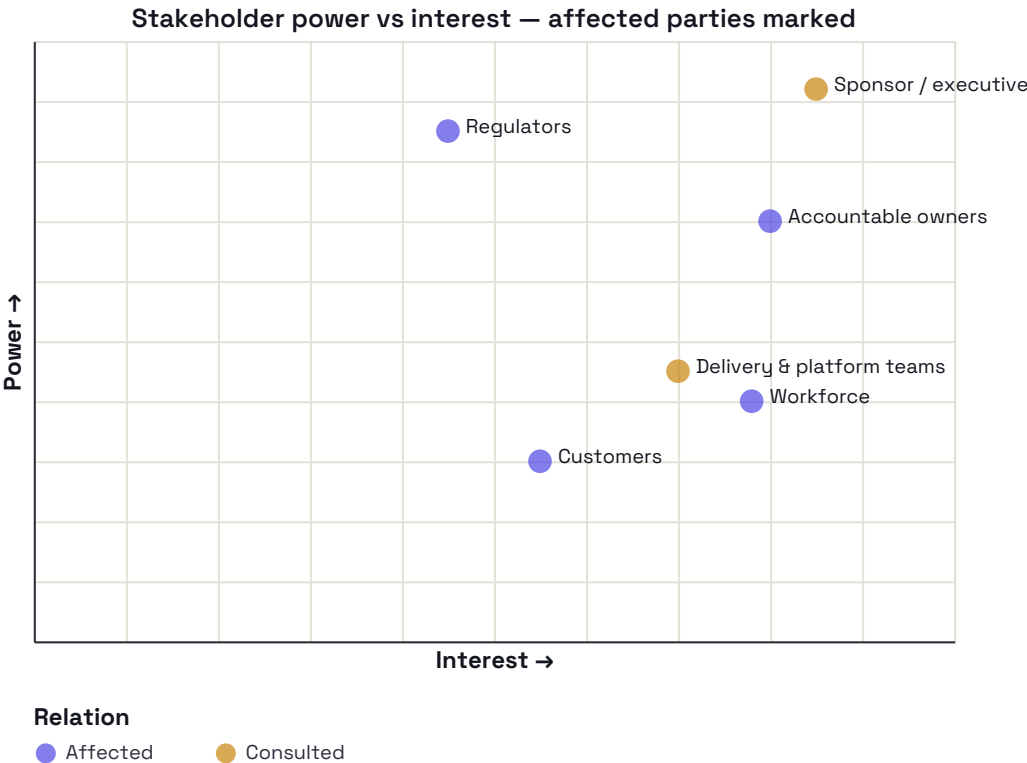


Figure 10.2 — Stakeholder power/interest, with affected parties marked. The grid is the classical tool; the addition is that four of the six parties shown are *affected by autonomy*, not merely consulted about it. The workforce sits high on interest and is easy to under-power; regulators sit high on power and are easy to under-interest. Both are mistakes the agentic enterprise punishes later.

A note on the new actor class. AEAf treats agents as **actors**, not stakeholders — an agent performs work and holds no stake or viewpoint of its own. The agent appears in Vision as the *subject* of the ambition and appetite, and as a future performer of capabilities, but the stakeholders are the humans and entities the agent's work affects. Conflating the two — speaking of "the agent's interests" — is a category error the actor model (→ see AEAf Framework §7) exists to prevent.

10.4 Vision-phase inputs and outputs

The table below sets out what the Vision phase consumes and produces. The classical inputs and outputs are retained; the three rows marked **new** are the agentic-enterprise additions, and each is annotated with where it is later consumed.

DIRECTION	ITEM	CLASSICAL OR NEW	SOURCE / DOWNSTREAM CONSUMER
Input	Business strategy & drivers	Classical	The sponsor and strategy function.
Input	Stakeholder landscape	Classical (extended)	Now includes affected parties (§10.3).
Input	Architecture capability, principles, repository	Classical	Stood up in the Preliminary phase (§10.5).
Input	Autonomy-level scheme & accountability model	New	Established in the Preliminary phase; read here to set appetite.
Input	Regulatory & risk context	Classical (heavier)	Compliance and risk functions; shapes the consequence envelope.
Output	Statement of Architecture Work	Classical	The cycle's mandate; basis of the final gate.
Output	Architecture Vision	Classical	Sponsor-level target sketch; refined by later phases.
Output	Human-Agent Ambition Statement	New	Consumed by Phase 2 capability allocation (→ §11) as its north star.
Output	Risk/Autonomy Appetite (consequence envelope + appetite ceiling)	New	Bounds autonomy levels in Phase 6 (→ §15) and the governance gate (→ §20).
Output	Candidate high-value capabilities for allocation	New	The shortlist Phase 2 allocates across human / agent / hybrid performers.

DIRECTION	ITEM	CLASSICAL OR NEW	SOURCE / DOWNSTREAM CONSUMER
Output	Secured sponsor approval	Classical	Recorded authorisation to proceed.

The three new outputs are deliberately *candidates and ceilings*, not decisions. Vision does not allocate any capability to an agent, does not assign an autonomy level, and does not design a guardrail — those are the work of later phases. What Vision does is name the candidates worth allocating, declare the ambition that ranks them, and set the ceiling no allocation may exceed. This division keeps Vision at the right altitude and keeps the binding decisions where the evidence to make them lives.

10.5 What Vision assumes the Preliminary phase already produced

Vision does not stand up the architecture capability — it uses one. The Preliminary phase (AEAF Method Phase P) is where the capability and its standing instruments are established, and Vision reads four of them as prerequisites. Stating them here makes the dependency explicit and points to where each is produced and later used.

PRELIMINARY PREREQUISITE	WHAT IT PROVIDES VISION	PRODUCED IN	LATER USED / DETAILED IN
Architecture capability stood up	The team, repository, and governance structure that runs the cycle at all.	Preliminary	Book 2 — capability set-up (→ see AEAF in Practice §4).
Principles, including runtime ones	The principle catalog with the Enforcement attribute (advisory / design-time / runtime guardrail) that Vision honours and the gate will check.	Preliminary	Defined in Ch.8 (→ see AEAF Framework §8).

PRELIMINARY PREREQUISITE	WHAT IT PROVIDES VISION	PRODUCED IN	LATER USED / DETAILED IN
Autonomy-level scheme	The graded scale against which Vision expresses the autonomy appetite ceiling.	Preliminary	Levels assigned per capability in Ch.15 (→ see AEAF Framework §15).
Accountability model	The rule that every capability has exactly one named human owner, so Vision can name candidate owners for candidate capabilities.	Preliminary	Operationalised in the Trust & Accountability Matrix (→ see AEAF Framework §18 / §20).

The relationship is one of dependency, not duplication. The Preliminary phase establishes the *standing* instruments — the scheme, the model, the principles — that outlast any single cycle. Vision *applies* them to this cycle's drivers: it does not invent an autonomy scale, it sets this cycle's appetite against the existing one; it does not write the accountability rule, it names candidate owners under it. An enterprise that reaches Vision without these instruments has skipped the Preliminary phase and should return to it — setting an autonomy appetite with no scale to express it in, or naming an ambition with no principles to bound it, produces a vision that the later phases cannot act on.

10.6 A worked vision vignette

A mid-size insurer opens an architecture cycle. Its driver is concrete: claims handling is slow and inconsistent, and a competitor now settles simple motor claims same-day. The sponsor is the Chief Operating Officer.

Drivers and scope. The cycle is scoped to the *Settle a motor claim* capability and the value stream around it. Out of scope: complex liability claims, fraud investigation, and anything touching injury — named out explicitly so the later phases cannot quietly pull them in.

Human-agent ambition. Stated plainly: *the insurer wants agents to handle the assessment and settlement of low-value, documentary, no-injury motor claims end to end, so that human claims handlers are freed to focus on disputed, complex, and vulnerable-customer cases where judgment is*

the value. The ambition names what moves to agents (routine documentary settlement) and what stays human by intent (judgment cases, vulnerable customers) — it is a direction, not a design.

Risk/autonomy appetite. The consequence envelope is set by the executive: agents may settle claims **up to €2,000 without a human signature**; anything above routes to a human; no claim involving injury or a flagged vulnerable customer may be settled by an agent at any value. The autonomy appetite ceiling is **on-the-loop** for in-envelope settlement — a human monitors and can intervene, but does not approve each one — and the appetite explicitly forbids out-of-the-loop operation this cycle. These are ceilings; Phase 6 will set the actual level, and it may set it lower.

Candidate high-value capabilities for allocation. Vision shortlists three: *assess a documentary claim*, *calculate a settlement within policy*, and *issue a payment within limit*. It allocates none of them — that is Phase 2's work (→ see AEAF Framework §11) — but it names them as the candidates the ambition ranks highest and the appetite permits.

Stakeholders, including the affected. The COO sponsors. The named accountable owner for the capability is the Head of Motor Claims, who agrees in Vision to remain answerable for what the agents settle. The **workforce** — the claims handlers whose routine work moves to agents — is engaged here as an *affected* party, not informed after the fact, because the ambition redistributes their work. The **regulator** is mapped as a high-power affected party: the insurer must be able to answer, for any settled claim, who was accountable. **Customers** are affected by an agent that pays them and must always be able to reach a human.

Secured approval. The COO approves the Statement of Architecture Work, the ambition, and the appetite together — the approval explicitly covers the €2,000 envelope and the on-the-loop ceiling, so that no later phase can widen them without returning to a sponsor. The cycle proceeds with its bounds set before its first domain decision.

What this vignette shows is the discipline the chapter argues for: the two new outputs are decided by the accountable executive, in plain terms, *before* anyone designs a single agent — so that capability allocation, agent design, and governance all inherit a bound they did not get to set for themselves.

Vision ends with a sponsored mandate and the bounds that make the rest of the method governable. The next phase takes the candidate capabilities and decides, for each, who or what performs it and under what oversight — capability allocation and the automation frontier (→ see AEAF Framework §11). The per-capability autonomy levels that the appetite here only ceilings are assigned in

Intelligence & Agency architecture (→ see AEAF Framework §15), and the gate that checks all of it against the bounds set in Vision is continuous and runtime governance (→ see AEAF Framework §20).

Chapter 11 — Business & capability architecture

In brief

This chapter is the *how-to* for Phase 2 of the AEAF Method — the agentic counterpart to TOGAF's Phase B. Where TOGAF Phase B maps capabilities, value streams, processes, and organization on the assumption that people and deterministic systems do the work, AEAF Phase 2 carries one extra dimension through all of it: a **performer type** — human actor, agent actor, or hybrid team — and an autonomy level per capability. The phase produces the classical business-architecture outputs *plus* two signature AEAF deliverables: the **capability allocation** (who or what performs each capability, decided with the Ch.7 framework) and the **Capability Automation-Frontier Map** (the as-is and target boundary between human- and agent-performed work). This chapter does not re-derive the allocation framework, the autonomy scale, or oversight theory — those live in Ch.7 (→ see AEAF Framework §7) and the meta-model (→ see AEAF Framework §5). It shows how to *run the phase*: what goes in, what comes out, how value streams and processes are redesigned for mixed execution, what changes in the organization, and a worked capability-map excerpt with allocations and a target frontier.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

1. What Phase 2 produces, and what is new

Business architecture answers four questions about an enterprise: what it does (**capabilities**), how value flows to a customer (**value streams**), how that work is carried out step by step (**processes**), and who is arranged to do it (**organization**). Phase 2 develops the target state for all four, baselined against today and traced to the Vision (→ see AEAF Framework §10).

The classical four outputs survive intact. What changes is that each acquires a performer dimension, and two new outputs are added that have no Phase B analogue.

PHASE 2 OUTPUT	CLASSICAL (TOGAF PHASE B)	WHAT AEAFF ADDS
Capability model	Hierarchical, actor-agnostic map of what the enterprise can do.	Unchanged in structure — but each capability now carries a performer type and autonomy level attribute (Section 2).
Value-stream map	End-to-end value flow, stage by stage.	Performer type per stage; explicit agent insertion points and human handoff points (Section 4).
Process models	Step-level process architecture.	Steps tagged human / agent / hybrid; escalation and fallback paths drawn as first-class, not exceptions (Section 4).
Organization map	Units, roles, reporting structure.	Roles gain <code>permittedPerformerTypes</code> ; functions decouple from headcount; a supervision layer appears (Section 5).
Capability allocation (new)	—	The recorded decision of who/ what performs each capability, at what autonomy, under what oversight (Section 3).
Capability Automation-Frontier Map (new)	—	The as-is and target boundary between human- and agent-performed work, dated and with trajectory (Section 3).

The capability model stays **actor-agnostic** by design, exactly as in TOGAF: "Settle a refund" is the capability, and *who settles it* is a separate allocation decision. This separation is what lets work move between humans and agents without rewriting the capability model — the single most important structural reason AEAFF keeps the capability map as the spine of the phase.

2. Building the capability map with a performer dimension

The capability map is built the way it always was — top-down decomposition into a stable, outcome-named hierarchy, typically three to four levels (L0 enterprise, L1 domain, L2 capability, L3 sub-capability). AEAF changes nothing about that mechanics. It adds one column to the model and one decomposition discipline.

2.1 The added attributes

Every leaf-level capability in an AEAF capability model carries, beyond the classical maturity/importance assessment, two new attributes drawn from the meta-model (→ see AEAF Framework §5):

- **Performer type** — human actor, agent actor, or hybrid team — the *current* allocation.
- **Autonomy level** — for any capability an agent touches, the graded independence granted, from suggest-only through bounded action to autonomy-within-bounds (the Autonomy-Level Classification is defined in its home chapter; Phase 2 *uses* it).

These are not decoration. They are the hooks the allocation (Section 3) and the frontier map hang on, and they are what makes a capability model in the agentic enterprise legible as a workforce picture rather than only a strategy picture.

2.2 Decompose until the performer is uniform

The discipline AEAF adds to capability modelling is this: **decompose a capability until each leaf has a single sensible performer**. A capability is rarely uniform in its allocation. "Settle a customer refund" is part clear-cut high-volume work an agent can own and part contested high-value work that must stay human. Modelling it as one capability with one performer forces a false choice; decomposing it by the allocation factors (consequence, reversibility, ambiguity — Ch.7 §2.2) reveals the natural bands, each with its own clean allocation.

Phase 2 rule. If a capability would receive two different performer types or two different autonomy levels across the work it covers, it is not yet decomposed enough. Split it until each leaf allocates cleanly. The frontier sits inside capabilities, and it can only be drawn if the capabilities are fine-grained enough to hold a line.

This is the same fractal-depth judgement classical capability modelling already requires — AEAF just gives it a sharper stopping rule: decompose to the grain at which performer and autonomy are uniform.

3. The two signature outputs: allocation and the frontier map

Phase 2's distinguishing deliverables are the **capability allocation** and the **Capability Automation-Frontier Map**. The *decision logic* behind both — the six allocation factors, the conservative decision flow, the consequence × confidence oversight table — is the Ch.7 framework and is not repeated here (→ see AEAF Framework §7). What follows is how that framework **lands as a phase deliverable**: what you actually produce, in what order, with what evidence.

3.1 Producing the capability allocation

The allocation is generated by running the Ch.7 decision flow once per leaf capability and recording the result as a structured entry. The output is not a narrative; it is a register, one row per capability, that every later phase reads.

Each allocation entry records four interlocked decisions (Ch.7 §2.1) plus the evidence that justified them:

ALLOCATION FIELD	WHAT IT RECORDS	SOURCE
Performer	Human actor / agent actor / hybrid team.	Ch.7 decision flow (Fig. 7.1).
Autonomy level	The independence granted to any agent involved.	Autonomy-Level Classification.
Oversight mode	In-the-loop / on-the-loop / out-of-the-loop.	Ch.7 §4, Table 7.1.
Accountability owner	The single named human answerable for the outcome.	Mandatory for every agent-touched capability.
Basis	The factors that drove the decision and the assurance evidence (or its absence).	The phase's audit trail.

The allocation register feeds two artifacts catalogued in Ch.21: the **Human-Agent RACI** (the responsibility/accountability split per capability) and the **Trust & Accountability Matrix** (capability → owner → oversight → assurance evidence). Phase 2 produces the allocation; those artifacts are how it is recorded and carried forward.

3.2 Producing the automation-frontier map

The **Capability Automation-Frontier Map** is a view over the allocated capability model that shows, per capability, the current human/agent boundary and the direction it is planned to move. It is the aggregate picture the allocation register makes possible. Phase 2 produces *two* states of it.

FRONTIER STATE	WHAT IT SHOWS	WHY IT MATTERS
As-is frontier	Where the human/agent boundary sits today, per capability, dated.	The honest baseline. Without it, "target" has no anchor and progress cannot be measured.
Target frontier	Where the boundary is planned to sit at the horizon, per capability, with the trajectory and trigger for each move.	The transformation intent, expressed as concrete re-allocations rather than a headline automation percentage.

The frontier is **time-versioned, not a fixed split** (Ch.7 §3.1): every allocation carries an `effectiveFrom` date, so the map states a position *as of* a date and the gap between as-is and target is a set of dated, evidenced moves. Describing the frontier as "60% automated" discards its only useful property — its trajectory. Phase 2 records position *and* direction.

Two disciplines are binding when drawing the target frontier:

- **A target move is a hypothesis, not a commitment.** The target frontier states where the enterprise *intends* the boundary to move and on what assurance evidence the move depends. Whether a move actually happens is gated, at run time, on the assurance case (→ see AEAF Framework §17). Phase 2 sets the intent; it cannot pre-authorise an unassured move.
- **Frontiers retreat as well as advance.** A target frontier that only ever moves work toward agents is not being governed — it is being assumed safe. The map must be able to represent a planned *retreat* (a capability moving back toward human performance or tighter oversight) on a tightening regulation or a known-fragile eval (Ch.7 §3.3).

The reusable template for this artifact, with a filled example, is Book 2 territory (→ see AEAF in Practice §15); Section 6 shows an excerpt to make the shape concrete.

4. Value-stream and process redesign for mixed execution

Allocation decides *who* performs a capability. Value streams and processes decide *how the work flows* — and a flow that mixes human and agent performers is not the human flow with some steps automated. It has to be redesigned around the two things that determine whether a blended flow works: **where agents sit** and **where control passes between performer types**.

4.1 Where agents sit in a value stream

A value stream is a sequence of stages that turn a customer need into a delivered outcome. In a blended value stream, each stage carries a performer type, and the design question at each stage is whether an agent can own it, assist within it, or must leave it to a human. AEAF names three positions an agent can take in a stream:

AGENT POSITION IN A STREAM	WHAT THE AGENT DOES	TYPICAL STAGE
Owns the stage	The agent performs the whole stage within guardrails; humans oversee on- or out-of-the-loop.	High-volume, well-specified, reversible stages (triage, drafting, classification, routing).
Assists within the stage	The agent prepares, proposes, or summarises; a human decides and acts.	Judgment-heavy stages where the agent compresses the human’s work without taking the decision.
Absent from the stage	The agent does not perform; the stage is human, possibly informed by agent output upstream.	High-consequence, contested, or legally human-mandated stages.

The redesign discipline is to look at the value stream as a whole and place the agents where they shorten the stream without putting an unassured agent on a high-consequence stage. The frontier, drawn at the capability level in Section 3, becomes visible in the value stream as the set of stages where the performer type changes.

4.2 Handoffs are the load-bearing design

The points where a value stream changes performer type — agent to human and back — are **handoff points**, and they are where blended flows succeed or fail. A handoff is not an exception to the flow; it is part of the flow, and AEAF requires it to be designed explicitly, not left to emerge.

The handoff and escalation patterns themselves — escalation, delegation, override, resumption, and the shared-context requirement that makes them work — are defined in Ch.7 §5 and not repeated here. The Phase 2 obligation is to **draw them into the process model as first-class paths**:

- Every agent-owned step has a defined **escalation path** — the trigger, the target human, and the fallback if no human is available (hold, queue, or safe default).
- Every handoff transfers **full context**, so the receiving performer is not starting from a fresh ticket (Ch.7 §5.1).
- The **fallback path** for a failed or low-confidence agent step is drawn as a normal branch, not an error state — because for a probabilistic performer, "the agent was unsure" is an ordinary, expected outcome, not a defect.

A process model for a blended capability that shows only the happy path is incomplete. The escalation and fallback paths are where the architecture earns its keep.

4.3 A blended value stream with agent insertion points

Figure 11.1 shows a supplier-onboarding value stream redesigned for mixed execution. The colour convention is binding across AEAF (→ see AEAF Framework §5): **amber** is human judgment, **indigo** is agent intelligence, and a **split** node is a hybrid stage where an agent acts and a human stays on- or in-the-loop.

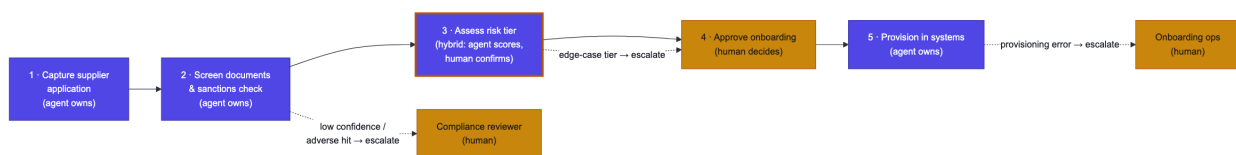


Figure 11.1 — A supplier-onboarding value stream redesigned for mixed execution. Agents own the high-volume, reversible stages (capture, screening, provisioning, all indigo); a hybrid stage scores risk with an agent and confirms with a human (split); the approval stage stays human because the consequence is high and the decision is contested (amber). The dotted edges are the designed escalation paths — they are part of the flow, not error states. The two performer-type changes (stage 2→3 and 3→4) are exactly where this value stream crosses the automation frontier.

5. Organizational impact

When a role can be filled by a human, an agent, or a hybrid team, the organization map stops describing a population of people and starts describing a population of *performers*. Phase 2 records the structural consequences; it does **not** plan the workforce transition — that sequencing, the reskilling, and the role-change choreography are deferred to migration planning (→ see AEAF Framework §16).

Three structural changes are recorded in the Phase 2 organization map (the underlying mechanics are Ch.7 §1):

ORGANIZATIONAL CHANGE	WHAT IT MEANS IN THE ORG MAP	WHAT PHASE 2 RECORDS
Roles carry permittedPerformerTypes	A role is no longer a synonym for a person; "Claims triage officer" may be human, agent, or hybrid over time.	The performer types each role permits, and its current allocation.
Functions decouple from headcount	A function is sized by the work, not by how many people it would take, because agents may perform some or all of it.	The function, its capabilities, and the performer mix — not a headcount.
A supervision layer appears	Where work is agent-performed, the human role shifts from doing to overseeing, handling exceptions, and owning outcomes.	The new oversight and accountability roles the allocation creates.

The structural pattern is worth stating plainly because it is the one most often missed: an agent allocation does not shrink the old org chart, it **reshapes** it. A team of fifteen processors and one supervisor does not become ten processors and one supervisor; it becomes a fleet of agents, three exception-handlers, and one accountability owner. The new structure has fewer doers and more overseers, and it needs roles — eval owner, agent owner, exception handler — that the old structure did not have.

The transition implication, flagged and deferred. Every frontier move in the target map implies people whose work shifts from doing to overseeing, or to a different capability, or out of the function. Phase 2's duty is to make that implication visible in the org map — to surface, capability by capability, where the human work changes. The duty to plan the transition humanely — sequencing, reskilling, redeployment, the change-management of it — belongs to migration planning (→ see AEAF Framework §16) and is not discharged here. Naming the implication without planning the transition is the correct Phase 2 boundary.

6. Phase 2 inputs and outputs

Phase 2 sits between Vision (→ see AEAF Framework §10) and Data & Knowledge architecture (→ see AEAF Framework §12), and it reads from and writes to the method's centre, Intent & Assurance Management (→ see AEAF Framework §17), exactly as every TOGAF phase touched Requirements Management.

DIRECTION	ITEM	FROM / TO	NOTES
Input	Architecture Vision, scope, human-agent ambition, risk/consequence envelope	Phase 1 (Vision)	Sets which capabilities are in scope and the risk appetite that bounds allocation.
Input	Baseline business architecture (capabilities, value streams, processes, org)	Existing repository	The as-is to baseline against; may be thin in a greenfield.
Input	Agentic principles and the autonomy-level scheme	Phase P (Preliminary)	The principles (some runtime-enforced) and the autonomy scale Phase 2 applies.

DIRECTION	ITEM	FROM / TO	NOTES
Input	Declared intents and consequence/ reversibility data per capability	Intent & Assurance centre	Feeds the allocation factors (Ch.7 §2.2).
Output	Target capability model with performer type + autonomy per capability	→ Phases 3–6, repository	The actor-agnostic spine, now carrying the performer dimension.
Output	Capability allocation register	→ Phases 6–9, Human-Agent RACI, Trust & Accountability Matrix	The signature decision record (Section 3.1).
Output	Capability Automation-Frontier Map (as-is + target)	→ Phase 8 (migration), governance	The signature frontier deliverable (Section 3.2).
Output	Blended value-stream and process models	→ Phases 4–6	Agent insertion points, handoffs, fallback paths (Section 4).
Output	Organization map with permittedPerformer Types and the supervision layer	→ Phase 8 (transition planning)	Surfaces the workforce implication; does not plan it (Section 5).
Output	Capability gaps and candidate frontier moves	→ Phase 7 (Opportunities & Solutions)	The work packages the next phase sequences by trust earned.

The two new outputs in bold are what make this Phase 2 rather than TOGAF Phase B. Everything else is the classical phase carrying one extra dimension.

7. Worked capability-map excerpt with allocations and a target frontier

A short worked excerpt makes the phase concrete. The example is a fictional B2B SaaS company's **Customer Operations** L1 domain; figures are chosen to make the mechanics legible, not drawn from a specific organization. (Ch.7's worked example follows a *single* capability — refund settlement — to its depth; this excerpt instead shows a *slice of the map*, several capabilities side by side, to show the frontier as an aggregate.)

7.1 The allocation register excerpt

Each L2 capability has been decomposed (Section 2.2) until its performer is uniform, run through the Ch.7 decision flow, and recorded.

L2 CAPABILITY	AS-IS PERFORMER	TARGET PERFORMER	AUTONOMY (TARGET)	OVERSIGHT (TARGET)	ACCOUNTABILITY OWNER	MOVE TRIGGER
Triage support ticket	Hybrid (agent drafts, human routes)	Agent actor	Autonomy within bounds	Out-of-the-loop, sampled review	Support Ops Lead	Eval error rate < 2% held 1 quarter
Answer Tier-1 query	Human actor	Hybrid team	Bounded action	On-the-loop	Support Ops Lead	Knowledge-corpus freshness signed off
Process refund ≤ EUR 500	Human actor	Agent actor	Autonomy within bounds	Out-of-the-loop, spend cap	Finance Ops Lead	Assurance case approved
Handle escalated complaint	Human actor	Human actor (agent assists)	Suggest-only	In-the-loop	Customer Care Lead	None — stays human
Approve goodwill credit > EUR 500	Human actor	Human actor (agent assists)	Suggest-only	In-the-loop	Finance Ops Lead	None — legally/contractually human

Two capabilities are deliberately marked "stays human" — the register is not a plan to automate everything, and a target frontier that left no work with humans would be a warning sign, not an achievement.

7.2 The target frontier, drawn over the map

Figure 11.2 plots the same five capabilities as a frontier excerpt: where each sits today and where the target moves it. Amber is human-performed, indigo is agent-performed, split is hybrid; the dotted arrows are the *planned, assurance-gated* moves.

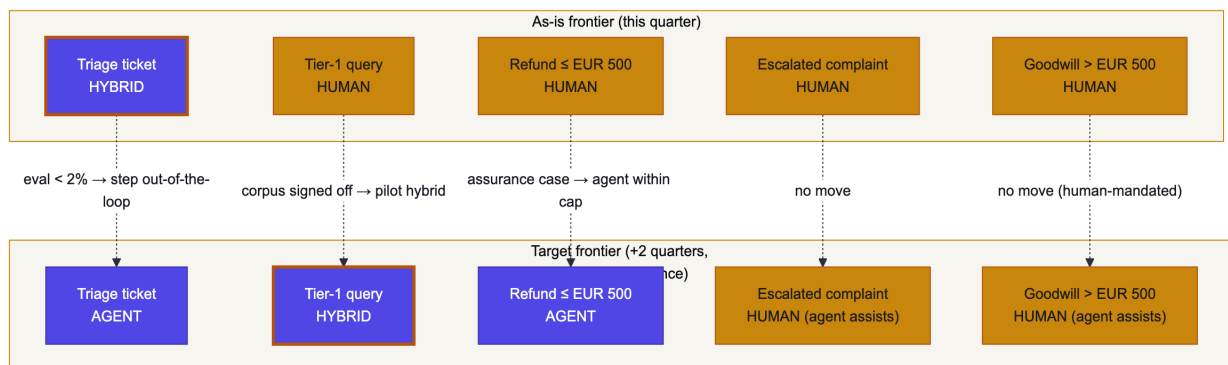


Figure 11.2 — As-is and target automation frontier for the Customer Operations slice.

Three capabilities are planned to move toward agent performance (trriage hybrid→agent, Tier-1 human→hybrid, refund human→agent), each with a named, assurance-gated trigger; two stay human because the work is contested or human-mandated. The dotted arrows are *intended* moves, not authorised ones — each fires only when its assurance case holds (→ see AEAF Framework §17). The gap between the two columns is the transformation, expressed as concrete re-allocations rather than a headline automation figure — and it is what migration planning (→ see AEAF Framework §16) will sequence.

7.3 What the excerpt demonstrates

The excerpt shows the phase working as a whole: capabilities decomposed to a uniform performer; each allocation a four-part decision recorded in a register; the frontier readable in two dated states with a trajectory between them; some capabilities held human on purpose; and every move gated on assurance rather than asserted. This is Phase 2's output — the actor-agnostic capability map turned into a governed, time-versioned picture of who does what in the blended workforce, ready for the data, application, technology, and Intelligence & Agency phases that realize it.

8. What this chapter hands forward

Phase 2 produces the workforce picture the rest of the method builds on.

- The **allocation framework, autonomy scale, and oversight theory** this phase applies are defined in the reworked Actor model (→ see AEAF Framework §7); Phase 2 *executes* them as deliverables, it does not re-derive them.
- The **formal entities** behind every allocation entry — Capability, Capability Allocation, Autonomy Level, Oversight Control, Accountability Owner, Automation Frontier — are in the meta-model (→ see AEAF Framework §5).
- The **frontier moves and the human transition** that the target map implies are planned and sequenced in opportunities & migration (→ see AEAF Framework §16); Phase 2 surfaces the implication and stops there.
- The **assurance** that gates every target frontier move from intent to authorisation is Intent & Assurance management (→ see AEAF Framework §17).
- The **reusable template** for the Capability Automation-Frontier Map, with a fully filled example, is in the artifact template set (→ see AEAF in Practice §15).

Chapter 12 — Data & knowledge architecture

In brief

Phase 3 of the AEAF Method is data architecture, and the agentic enterprise needs all of it: conceptual, logical, and physical models; master data; governance, classification, lineage. None of that retires when agents arrive — a probabilistic actor grounded on ungoverned data is worse than a deterministic system, not better, because it will assert ungoverned content with fluent confidence. What the agentic enterprise *adds* is a second layer on top of the transactional store: the **cognitive substrate** — data deliberately curated as material for machine cognition. Knowledge corpora, embeddings and vector indexes, retrieval (RAG) corpora, agent memory, and knowledge graphs are not implementation detail; they are architecture objects with their own freshness, classification, and access semantics. This chapter draws the **Data ↔ Intelligence boundary** — the point at which a governed Data Entity becomes a Knowledge Corpus an agent reads from — and states what governance must travel across it, because the substrate that feeds machine reasoning is architected, not improvised. The agent that *uses* the corpus is designed in Chapter 15; how retrieval quality is *evaluated* is Chapter 17; *which* vector database to choose is a Book 2 decision. This chapter governs the substrate itself.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where this chapter contrasts the two, it does so to locate AEAF's additions to the data domain precisely, not to characterise TOGAF.

12.1 Classical data architecture does not retire

The data domain in AEAF is the data domain you already know, retained in full. AEAF's position is the opposite of replacement: the more an enterprise grounds agents on its data, the more the classical disciplines matter, because every weakness in the data is amplified by a confident generator on top of it.

The four classical pillars all stay, with their meaning unchanged.

PILLAR	WHAT IT DOES	WHY THE AGENTIC ENTERPRISE NEEDS IT MORE, NOT LESS
Conceptual / logical / physical models	Define what the enterprise's information means, how it is structured, and how it is physically stored.	A retrieval corpus inherits the meaning of its source. If "customer" is ambiguous in the model, it is ambiguous in everything the agent retrieves and asserts.
Master data management (MDM)	Maintain one authoritative version of core entities — customer, product, supplier.	An agent that retrieves three conflicting addresses for one customer has no basis to choose. MDM is what makes a corpus answerable.
Data governance	Assign ownership, quality rules, and stewardship to data.	The substrate needs an owner for every corpus; that owner is a data owner, governed by the same regime, not a new one invented for AI.
Lineage	Record where data came from and how it was transformed.	Lineage is the precondition for provenance (§12.5) — an agent's answer is only traceable if the corpus chunk traces to a source.

The structural change AEAF makes is small and exact: the **Data Entity** gains one relationship — it may *feed* a Knowledge Corpus (→ see AEAF Framework §5). It does not change what a Data Entity is. The transactional record of an order is still a Data Entity governed by the order's data owner. What is new is that this entity can now be curated into a substrate for cognition, and that act of curation is the subject of the rest of this chapter.

The two-layer rule. Transactional data answers what is true now and is the system of record. The cognitive substrate answers what an agent may read to reason and is a derived, governed projection of the record. Never let the substrate become the system of record — it is a copy optimised for retrieval, not the authoritative source.

12.2 The new layer: the cognitive substrate

The **cognitive substrate** is data curated as material for machine cognition. It is the part of the data domain that exists only because agents read it. A transactional database is queried with exact predicates by deterministic code; the cognitive substrate is queried by *meaning* — an agent asks for what is relevant to a goal, and the substrate returns grounding content. That difference in access mode is why the substrate is a distinct architectural layer with its own design concerns.

Five substrate types make up the layer. Each has a different shape, a different freshness profile, and a different failure mode, so each is governed as its own kind of object.

SUBSTRATE TYPE	WHAT IT HOLDS	PRIMARY ACCESS MODE	CHARACTERISTIC FRESHNESS	CHARACTERISTIC FAILURE MODE
Knowledge corpus (RAG)	Curated documents an agent retrieves from to ground answers — policies, product docs, prior cases.	Semantic retrieval over chunks	Periodic re-index as sources change	Stale or wrong chunk retrieved and asserted as current
Embeddings / vector index	The numeric representation of corpus content that makes semantic retrieval possible.	Nearest-neighbour search	Re-embedded when the embedding model or chunking changes	Representation drift — old and new vectors no longer comparable

SUBSTRATE TYPE	WHAT IT HOLDS	PRIMARY ACCESS MODE	CHARACTERISTIC FRESHNESS	CHARACTERISTIC FAILURE MODE
Agent memory (short-term)	The working context of a single task or conversation — what the agent has seen so far.	Read/written within a session	Lives and dies with the task	Context overflow; the agent forgets within the task
Agent memory (long-term)	Facts, preferences, and outcomes an agent retains across tasks.	Retrieved by relevance across sessions	Accumulates; needs decay and review	Memory poisoning; a wrong fact persists and compounds
Knowledge graph	Entities and the typed relationships between them, queried by traversal.	Graph traversal from seed entities	Updated as relationships change	Missing or wrong edge yields a confident but false connection

These types are not alternatives; a mature agent draws on several at once. A support agent might retrieve from a RAG corpus of policy documents (knowledge corpus + vector index), recall the customer's prior tickets (long-term memory), traverse from the customer to their contracted products (knowledge graph), and hold the current conversation in working context (short-term memory). Chapter 15 designs how an agent *combines* these; Phase 3 architects each substrate so that it *can* be combined safely.

12.2.1 Why the substrate is architected, not improvised

The common failure is treating the substrate as a side effect of building an agent — a folder of PDFs pointed at a vector database, set up once and forgotten. That improvised substrate has no owner, no classification, no freshness policy, and no record of what is in it. When the agent asserts something wrong, no one can say where it came from or who is accountable for the source.

The substrate is architected for the same reasons the transactional store is: so that it has an owner, a classification, a known freshness, a known access policy, and a place in the catalogue. The artifact that records this is the **Knowledge-Corpus Catalog** (§12.7), and the act of putting a corpus in it is the difference between a substrate that is governed and one that is merely present.

12.3 The Data ↔ Intelligence boundary

The defining act of Phase 3 is deciding when a **Data Entity** (a Data-domain object) becomes a **Knowledge Corpus** (an Intelligence & Agency object an agent reads from), and ensuring the governance that applied to the source travels across that boundary. This is where the Data domain and the +I domain meet, and it is the single most error-prone seam in the agentic enterprise's data architecture.

The boundary is a *projection*, not a copy. A corpus is derived from one or more Data Entities by selection, chunking, and embedding. The danger is that the projection silently drops the governance the source carried — a document classified "internal" is chunked, embedded, and indexed into a corpus an externally-facing agent can read, and the classification did not come along for the ride.

AEAF's rule is that **four governance attributes must travel with every corpus, inherited from the most restrictive contributing source**. They do not get re-derived or relaxed at the boundary.

GOVERNANCE ATTRIBUTE TRAVELLING ACROSS THE BOUNDARY	WHAT IT MEANS FOR THE CORPUS	WHAT GOES WRONG IF IT IS DROPPED
Classification	The corpus is classified at the level of its most restrictive source chunk (e.g. one "confidential" document makes the corpus confidential unless that content is excluded).	An agent retrieves and surfaces content above the requester's clearance — a disclosure with no human in the path.
Freshness obligation	The corpus carries the source's update cadence; if the source is revised weekly, the corpus is stale after a week.	The agent grounds on superseded policy and asserts it as current.
Access policy	Who and what may read the corpus is bounded by who may read the sources; an agent inherits, never exceeds, the source's access scope.	The corpus becomes a back door around source-level access controls.

GOVERNANCE ATTRIBUTE TRAVELLING ACROSS THE BOUNDARY	WHAT IT MEANS FOR THE CORPUS	WHAT GOES WRONG IF IT IS DROPPED
PII / regulated content	Personal or regulated data in the source remains personal or regulated in the corpus, with the same lawful-basis and retention constraints.	Embedding “anonymises” nothing; PII sits in the vector store outside its retention and lawful-basis controls.

Embeddings do not launder governance. A common and dangerous assumption is that turning text into a vector strips its sensitivity. It does not. An embedding is a recoverable representation of its source content, and content retrieved from a vector index is shown to whatever reads it. Classification, PII status, and access scope attach to the meaning, and the meaning survives embedding intact.

The boundary diagram in §12.6 shows this as the amber gate between the Data domain and the substrate: nothing crosses it without its governance.

12.4 Architecture concerns unique to cognition

Beyond classical data quality, the cognitive substrate raises concerns that have no analogue in transactional data architecture. They arise because retrieval is by *meaning* and generation is *probabilistic*, so the substrate's design directly shapes whether an agent's answer is right. These are Phase-3 design decisions — they are settled when the substrate is architected, not patched later.

12.4.1 Chunking and representation

A corpus is not retrieved as whole documents; it is split into **chunks** — passages sized for retrieval — and each chunk is embedded and indexed independently. Chunking is an architecture decision because it sets the unit of meaning the agent can retrieve. Chunks too large dilute relevance and waste the agent's context; chunks too small sever the context a

passage needs to be understood. The standard recursive approach splits on structure (paragraph, then line, then sentence) with a small overlap so meaning is not cut at a boundary.¹

CHUNK STRATEGY	WHAT IT OPTIMISES	WHEN IT FITS
Fixed-size with overlap	Predictable, simple indexing	Uniform documents, prototypes
Structure-aware (recursive)	Preserves semantic boundaries (paragraphs, headings)	General-purpose corpora — the common default
Semantic (embedding-driven split)	Coherent units regardless of formatting	High-value, heterogeneous documents where retrieval quality justifies the cost

The general guidance is to keep chunks in the range that balances context against precision — large enough to stand alone, small enough to stay relevant — and to carry chunk metadata (source, section, date) so that a retrieved chunk can be cited.¹ AEAF treats the chunking and overlap policy as a recorded property of the corpus, not a hidden parameter of a script, because changing it changes what the agent can know.

12.4.2 Embedding-model choice as an architecture decision

The **embedding model** turns content into the vectors that make semantic retrieval possible. AEAF treats its selection as an architecture decision with long-lived consequences, not a tuning knob, for three reasons.

First, the embedding model determines retrieval quality — its ability to place related meanings near each other in vector space sets the ceiling on what retrieval can find.² Second, it sets the **dimensionality**, which drives storage and search cost across the whole corpus.² Third, and most consequential, **changing the embedding model invalidates the entire index**: vectors produced by a different model are not comparable, so a model change forces a full re-embed of every corpus that uses it. The choice is therefore a commitment, and AEAF records the embedding model as a first-class attribute of every corpus in the Knowledge-Corpus Catalog.

DECISION DIMENSION	WHAT IT TRADES OFF
Retrieval quality	A stronger model retrieves more relevant content but costs more per item embedded. ²
Dimensionality	More dimensions can mean higher quality but more storage and slower search; some models support reduced dimensions at a small quality cost. ²
Multilinguality	A corpus that spans languages needs a model that places equivalent meanings together across them.
Hosting / privacy	A hosted model is simpler; a self-hosted model keeps sensitive content in the enterprise's boundary. ²
Switching cost	Changing the model means re-embedding everything — a deliberate migration, not a config change.

The *how-to* of selecting a specific embedding model and vector store for a given workload is a Book 2 decision (→ see AEAF in Practice §10); Phase 3's job is to make the choice an explicit, recorded, governed decision rather than a default no one owns.

12.4.3 Retrieval quality and grounding versus hallucination

Grounding is the mechanism that makes an agent's answer traceable to retrieved content rather than invented. Its effectiveness is bounded by **retrieval quality** — if retrieval returns the wrong chunks, the agent grounds confidently on the wrong content, and a confident wrong answer is worse than a refusal. Retrieval quality is therefore an architecture concern of the substrate, even though its *measurement* belongs to evaluation (→ see AEAF Framework §17).

Several substrate-design techniques raise the quality ceiling, and the architecture records which a corpus uses.

TECHNIQUE	WHAT IT DOES	ARCHITECTURAL COST
Re-ranking	A second-stage model re-orders retrieved candidates so the most relevant rise to the top. ³	Added latency and cost on the top results only
Hybrid search	Combines semantic similarity with keyword matching, so exact entity names and codes are not missed. ³	A keyword index alongside the vector index
Query transformation	Rewrites or expands the query before retrieval to improve recall. ³	Extra model calls before retrieval
Graph augmentation	Adds related facts from a knowledge graph to the retrieved passages. ³	A maintained graph and traversal step

The point for Phase 3 is not to mandate a technique but to make the corpus's retrieval design explicit: an agent's grounding is only as good as the substrate it grounds on, so the substrate is where grounding quality is decided. Whether the achieved quality clears its threshold is the assurance question answered in Chapter 17.

12.4.4 Freshness and drift

A corpus is a projection of sources that change, so it is correct only for as long as it tracks them. Two distinct decays threaten it, and AEAF names them separately because they are fixed differently.

- **Corpus freshness** is whether the content still reflects its source. It decays when a source is revised and the corpus is not re-indexed. The fix is a freshness policy — a stated re-index cadence tied to the source's update rate, recorded in the catalog and monitored against it.
- **Representation drift** is whether the *vectors* are still comparable. It occurs when chunking or the embedding model changes for part of a corpus but not all of it, leaving old and new vectors in the same index that cannot be compared. The fix is to treat any change to chunking or embedding as a full re-embed of the affected corpus, never a partial one.

For long-term agent memory, a third decay applies: accumulated facts go stale or turn out wrong, and without a **decay-and-review** policy a poisoned memory persists and compounds across tasks. Memory is governed substrate, not free state.

12.5 Provenance for traceability

Provenance is the chain from an agent's assertion back to the source content that grounded it: *this answer came from this chunk, which came from this document, version, and date*. It is the substrate's contribution to the framework's traceability spine (→ see AEAF Framework §5), and it depends on classical lineage (§12.1) reaching all the way into the corpus.

Provenance is engineered into the substrate, not reconstructed afterwards. It requires that every chunk carries metadata identifying its source document, section, version, and date, and that retrieval returns this metadata alongside the content so the agent can cite it.¹ A corpus without per-chunk provenance can ground an agent but cannot explain it — and an agent that cannot show where its answer came from cannot be assured, audited, or trusted in a consequential decision.

Provenance is the precondition for assurance. An agent's answer that cannot be traced to a source cannot be checked, so it cannot clear an eval gate or survive an incident review. Provenance is what makes the cognitive substrate accountable rather than merely capable.

12.6 Reference picture: data → corpus → retrieval → agent

The flow below is the canonical Phase-3 picture: governed transactional data is projected across the Data ↔ Intelligence boundary into a cognitive substrate, an agent retrieves from that substrate to ground its reasoning, and an accountability owner governs the boundary and the corpus. Indigo nodes are machine cognition and the substrate that feeds it; amber nodes are the human-governed gates and owners that bound it.

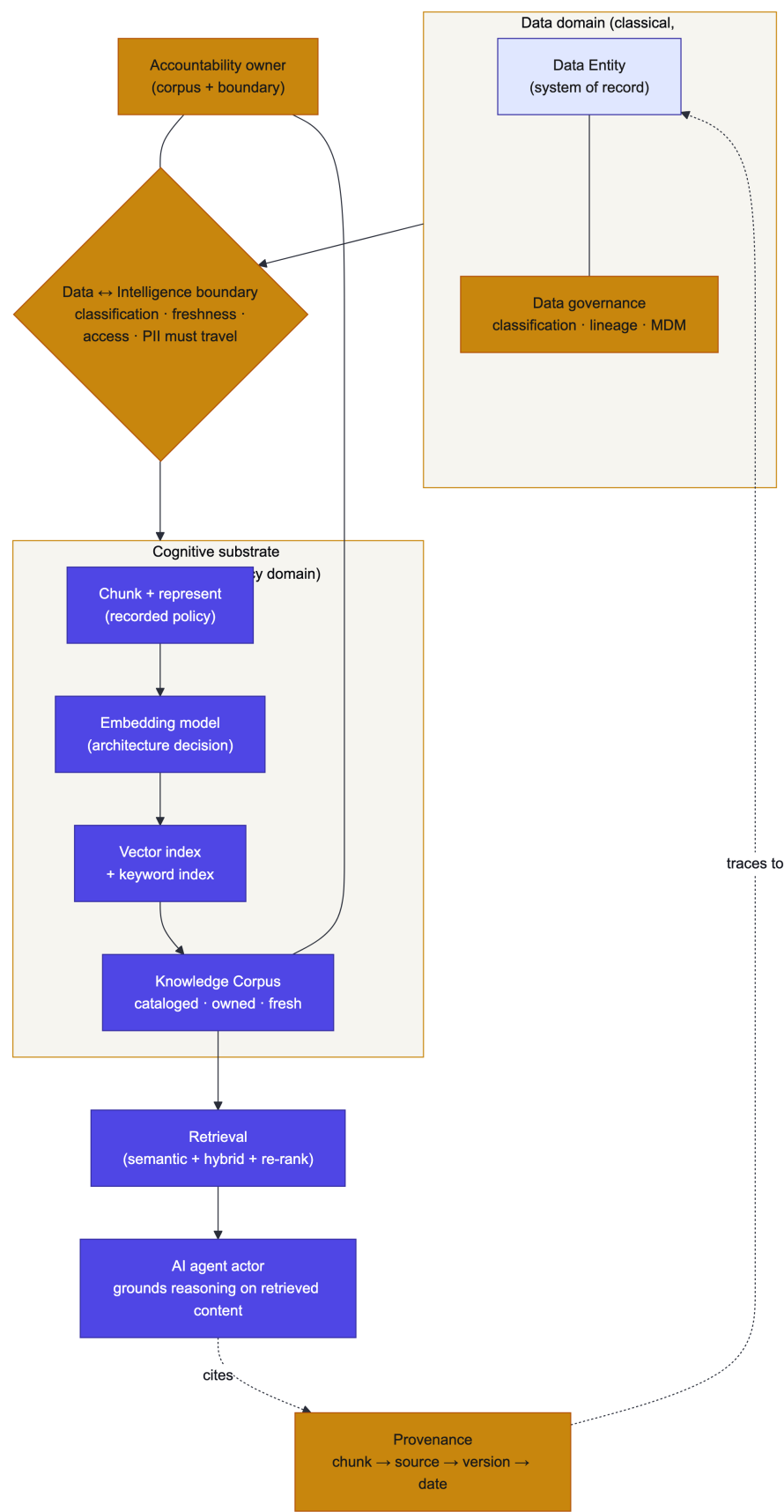


Figure 12.1 — The data → corpus → retrieval → agent flow. Transactional data (the system of record) is projected across the amber Data ↔ Intelligence boundary, where classification, freshness, access, and PII status must travel with it. In the substrate (indigo), content is chunked, embedded, and indexed into a cataloged, owned Knowledge Corpus. An agent retrieves from the corpus to ground its reasoning and cites the provenance chain back to the source Data Entity. The accountability owner (amber) governs both the boundary and the corpus. The substrate never becomes the system of record; it is a governed projection of it.

12.7 Governing the cognitive substrate

Governing the substrate answers one operative question: **what is each agent allowed to read?** That question is answered per corpus, recorded in the **Knowledge-Corpus Catalog**, and enforced as a guardrail. Substrate governance is therefore not a separate regime bolted onto data governance — it extends data governance with the access semantics that probabilistic, retrieving actors require, and it connects directly to the guardrail layer that bounds those actors at runtime.

12.7.1 Corpus classification and the read permission

Every corpus carries a classification (inherited per §12.3), an owner, and an explicit list of which agents may read it. The classification is not advisory metadata; it compiles into a **guardrail** — a machine-enforceable constraint on which agent may retrieve from which corpus, checked at retrieval time (→ see AEAF Framework §8 for the principle-to-guardrail compilation). An agent's permitted corpora are part of its definition, and an agent retrieving from a corpus it is not cleared for is a guardrail veto, not a quiet success.

This is the link between Phase 3 and the runtime assurance loop: the substrate decides *what content exists and how it is classified*; the guardrail decides *whether this agent may read it now*. The two are designed together — a classification with no enforcing guardrail is a label no one checks.

12.7.2 Data governance versus knowledge-corpus governance

The two governance regimes share a lineage but differ in what they protect against. Classical data governance protects the *correctness and integrity of the record*; corpus governance adds protection against *what a retrieving agent may surface and assert*. The contrast below is the substance of the Data ↔ Intelligence boundary.

GOVERNANCE DIMENSION	TRANSACTIONAL DATA ENTITY	KNOWLEDGE CORPUS
Primary question	Is the record correct and authoritative?	What may an agent read, and what may it surface from this?
Accessed by	Deterministic code with exact predicates	An agent retrieving by meaning
Classification role	Controls who may query the record	Compiles into a retrieval guardrail bounding which agent may read it
Freshness	Updated transactionally; correct now by construction	Decays after each source change until re-indexed; needs a stated cadence
Integrity threat	Bad write, lost update, inconsistency	Stale chunk, poisoned memory, representation drift, wrong chunk asserted
PII handling	Lawful basis, retention, subject rights on the record	Same constraints, plus the vector representation and any cached retrieval
Lineage / provenance	Where the data came from	Per-chunk provenance so an agent's answer can be cited and audited
Owner	Data owner / steward	Corpus owner (a data owner's remit, extended), recorded in the Catalog
Enforcement	Database access controls	Retrieval guardrails + access policy + freshness monitor

The Knowledge-Corpus Catalog is where this governance is made checkable: it is the register of every corpus an agent may read, and a corpus that is not in it is a corpus no one is accountable for.

12.8 Phase 3 inputs, outputs, and the Knowledge-Corpus Catalog

Phase 3 takes the business and capability architecture (which capabilities are allocated to agents, and therefore which need grounding) and produces both the classical data architecture and the cognitive substrate that those agents will read from. Its signature new output is the **Knowledge-Corpus Catalog**.

PHASE 3 INPUTS		SOURCE
Capability allocation & automation frontier (which capabilities are agent-performed)		Phase 2 (→ see AEAF Framework §11)
Intent statements for agent-performed capabilities (what grounding the outcome needs)		Intent & Assurance centre (→ see AEAF Framework §17)
Existing data architecture (conceptual/logical/physical models, MDM, classification scheme)		The enterprise's current Data domain
Source content inventory and lineage (candidate sources for corpora)		Data governance
Architecture principles with their Enforcement attribute (esp. data-classification and access principles)		Phase Preliminary / Ch.8 (→ see AEAF Framework §8)

PHASE 3 OUTPUTS	WHAT IT RECORDS	CARRIES FORWARD TO
Conceptual / logical / physical data models	The enterprise's information, structure, and storage (classical, retained)	All downstream phases
Updated classification, MDM, and lineage	Governance of the record, extended to reach into corpora	§12.3 boundary; §12.7 governance
Cognitive substrate design	Which corpora, memory stores, and graphs exist; their substrate type and shape	Phase 6 — agent design (→ see AEAF Framework §15)

PHASE 3 OUTPUTS	WHAT IT RECORDS	CARRIES FORWARD TO
Knowledge-Corpus Catalog	Each corpus: sources, classification, freshness policy, embedding model, chunking policy, access policy, owner, and the agents that read it	Phase 6, runtime guardrails, assurance
Data ↔ Intelligence boundary decisions	Which Data Entities feed which corpora, and the governance that travels	Guardrail/Policy Catalog (→ see AEAF Framework §8)
Retrieval-design record per corpus	Chunking, hybrid/re-rank/graph augmentation in use	Eval & assurance (→ see AEAF Framework §17)

The **Knowledge-Corpus Catalog** is one of the nine new AEAF artifacts (→ see AEAF Framework §21 for the full catalog; the worked template is in AEAF in Practice §15). In Phase 3 it is the deliverable that turns an improvised pile of indexed documents into a governed substrate: a register in which every corpus an agent may read has a named owner, an inherited classification, a freshness obligation, a recorded embedding and chunking policy, and an explicit list of the agents permitted to retrieve from it. Its columns are exactly the attributes that must be known for a corpus to be governed; an attribute the Catalog cannot fill is a corpus that is not yet ready to be read.

The Phase-3 test. A corpus is ready when its catalog row is complete: it has an owner, a classification inherited from its sources, a freshness policy tied to those sources, a recorded embedding and chunking policy, per-chunk provenance, and a guardrail that enforces which agents may read it. A corpus an agent reads but the Catalog does not list is the exact failure this phase exists to prevent.

12.9 What Phase 3 hands on

Phase 3 leaves the enterprise with two governed layers: the transactional record it always needed, and the cognitive substrate the blended workforce reads to reason. It does not design the agents — that is Phase 6, where an agent's choice of corpora, its memory strategy, and its retrieval behaviour are settled (→ see AEAF Framework §15). It does not measure

whether retrieval is good enough — that is the eval and assurance work of Chapter 17. And it does not pick the vector database or embedding model for a given workload — that is a Book 2 implementation decision (→ see AEAF in Practice §10).

What Phase 3 guarantees is narrower and load-bearing: that every corpus an agent will read is cataloged, classified, owned, fresh against a stated cadence, traceable to its sources, and bounded by a guardrail that says which agent may read it. The substrate that feeds machine cognition is architected — and the Data ↔ Intelligence boundary is the seam where the enterprise's data governance is extended to hold, rather than abandoned at, the point where data becomes the material an agent reasons over.

-
1. Chunking strategy, overlap, chunk-size guidance, and per-chunk metadata for citation follow standard RAG document-processing practice. Internal reference: *RAG Architecture Patterns — Knowledge Base* (RAG-PATTERNS-KB-001, v1.0, 2025-10-02), §"Document Processing & Chunking". [↔↔↔](#)
 2. Embedding-model trade-offs — retrieval quality, dimensionality and storage cost, multilinguality, hosting/privacy, and reduced-dimension options — per *RAG Architecture Patterns — Knowledge Base* (RAG-PATTERNS-KB-001, v1.0, 2025-10-02), §"Embedding Strategies & Optimization". Vector-store characteristics (index types, hybrid search, scale) per *Vector Database Comparison* (VECTOR-DB-COMP-001, v1.0, 2025-10-02). [↔↔↔↔↔](#)
 3. Re-ranking, hybrid search, query transformation, and graph augmentation as retrieval-quality techniques per *RAG Architecture Patterns — Knowledge Base* (RAG-PATTERNS-KB-001, v1.0, 2025-10-02), §"Advanced RAG Patterns" and §"Retrieval Optimization & Evaluation". [↔↔↔↔](#)

Chapter 13 — Application architecture in a non-deterministic world

In brief

Classical application architecture rests on one assumption: a component does the same thing every time. Given the same input, a function returns the same output, an integration contract is honoured exactly, and a test that passes today passes tomorrow. That assumption is what makes integration, testing, service-level agreements, and error handling work the way they do. A probabilistic component — anything driven by a model — breaks it. The same input can produce different output; the output can be well-formed and wrong; failure is a distribution, not an exception. This chapter is AEAF Method Phase 4 (≈ ADM C, Application). It does **not** redesign the application portfolio from scratch. It states the one new design problem — non-determinism crossing a component boundary — and the pattern that contains it: a **deterministic shell wrapping a probabilistic core**, with output contracts, guardrails at the boundary, fallback, and a human-handoff path. It places the **agent runtime** as an Application Component that *hosts* an agent actor (the component is not the actor), updates the integration patterns (model gateway, tool/function calling, the Model Context Protocol, event-driven async work) for AI, and defines what observability means when you must trace a decision, not just a request. Agent internal design, orchestration, and autonomy levels are Phase 6 (→ see AEAF Framework §15); the AI platform beneath the application is Phase 5 (→ see AEAF Framework §14); evaluation method is the centre (→ see AEAF Framework §17).

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where this chapter contrasts the two, it does so to locate AEAF's additions precisely, not to characterise TOGAF.

How to read this chapter: §1 states the core problem and exactly which classical assumptions break. §2 is the design discipline — the tactics for building reliable systems out of unreliable parts. §3 places the agent runtime in the metamodel as a component that hosts an actor. §4 updates the integration patterns. §5 defines observability for probabilistic components. §6 describes the application portfolio of the agentic enterprise and its relationship to the model and agent portfolios. The reference pattern — the deterministic shell around a probabilistic core — appears as Figure 13.1 in §2 and is referred to throughout.

1. The assumption that breaks, and what it takes down with it

1.1 The deterministic contract

Application architecture as practised assumes **specifiable behaviour**. A component publishes a contract — an interface, a schema, a set of pre- and post-conditions — and honours it deterministically. Where the term **determinism spectrum** is used here, it means the range from fully deterministic behaviour (same input → same output, specifiable and testable) to non-deterministic behaviour (probabilistic, bounded and evaluated rather than specified). Classical components sit at the deterministic end. Four pillars of the discipline rest on that position:

- **Integration** assumes a callee that returns a value of a known shape, or a known error. The caller codes against the contract, not against a probability.
- **Testing** assumes that a finite set of cases, once passing, certifies the component. Re-running the suite reproduces the result.
- **Service-level agreements** assume that "correct" is a property the component either has or lacks, so availability and latency are the variables worth measuring.
- **Error handling** assumes failure is the exception: a thrown error, a timeout, a non-2xx code — a discrete, detectable event.

1.2 What a probabilistic component does to each pillar

A model-driven component does not honour a deterministic contract. It produces a *plausible* output drawn from a distribution. Table 13.1 sets out what each classical pillar assumed, how a probabilistic component violates it, and the architectural consequence. This table is the problem statement for the whole chapter; everything in §2 answers a row of it.

Table 13.1 — How probabilistic components break the four pillars of application architecture.

PILLAR	CLASSICAL ASSUMPTION	WHAT A PROBABILISTIC COMPONENT DOES	ARCHITECTURAL CONSEQUENCE
Integration	Callee returns a value of known shape, or a known error.	Returns free-form or near-structured output that may not parse, may omit fields, may invent fields.	The boundary needs an output contract the caller can reject against — not a hope that the model complied.
Testing	A finite passing suite certifies the component.	Same input yields varying output; a case passing once does not certify the case.	One-time verification is replaced by continuous evaluation against thresholds (→ see AEA Framework §17).
SLA	“Correct” is binary; measure availability and latency.	Output can be well-formed, on time, and wrong. Quality is a distribution.	The SLA must add a quality dimension (acceptable-resolution rate, escalation rate), not only uptime.
Error handling	Failure is a discrete, detectable event.	The worst failure is confident and undetected — a fluent wrong answer with no error raised.	Detection must be active (validation, guardrails, confidence checks), not passive (catch the exception).

The fourth row is the dangerous one. A crashed component announces itself. A hallucinating component does not. Application architecture in a non-deterministic world is, before anything else, the discipline of making silent failure loud at a boundary you control.

1.3 The reframe: do not try to make the core deterministic

The instinct is to suppress the variance — pin the temperature to zero, over-constrain the prompt, demand the model "always" comply. This narrows the distribution; it does not make the component deterministic, and it trades away the capability you adopted the model for. AEAF takes the opposite stance: **accept that the core is probabilistic and engineer determinism at the boundary around it**. The system as a whole becomes predictable not because its core is, but because a deterministic shell decides what to do with whatever the core produces. §2 is that shell.

2. Designing for non-determinism

2.1 The deterministic shell around a probabilistic core

The governing pattern of this chapter is a **deterministic shell wrapping a probabilistic core**. The core is the model call (or the agent that runs on it) — non-deterministic, valuable, untrusted. The shell is ordinary, testable, deterministic code that surrounds the core and owns every interaction the rest of the system has with it. Nothing outside the shell talks to the model directly. The shell validates what goes in, checks what comes out against an output contract, enforces guardrails, and decides — deterministically — whether to return the result, retry, fall back, degrade, or hand off to a human. The principle is that **the system's contract is the shell's contract, not the model's**. The shell is specifiable and testable in the classical sense; the unpredictability is quarantined inside it.

Figure 13.1 shows the pattern. Indigo nodes are the probabilistic / agent core and the machine-enforced guardrails; amber nodes are the human-judgment path (the accountability owner and the handoff target). The shell — input validation, output-contract check, the routing decision, retry, and fallback — is deterministic code drawn in neutral grey.

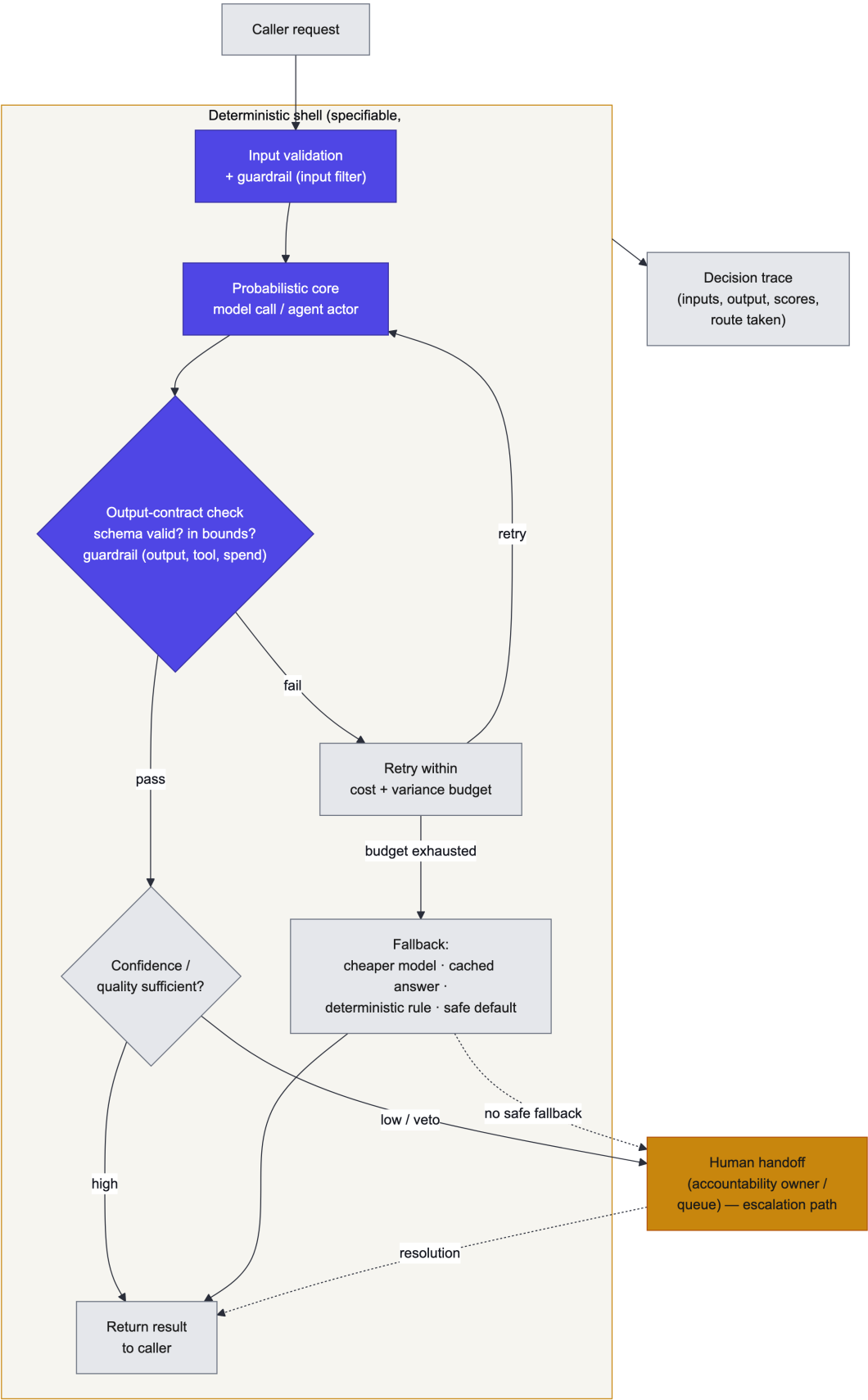


Figure 13.1 — The deterministic shell around a probabilistic core. The caller only ever sees the shell's contract. The probabilistic core (indigo) is bounded on both sides: an input guardrail before it and an output-contract check after it. The deterministic routing logic (grey) decides among return, retry, and fallback; low confidence or a guardrail veto takes the amber escalation path to a human. Every path writes a decision trace (§5). The pattern is referred to throughout this chapter as "the shell."

2.2 The design tactics

The shell is built from a small set of tactics. Each answers a specific way a probabilistic core misbehaves. Table 13.2 is the working catalogue: the tactic, the failure it addresses, and how it is realised at the boundary. These are the building blocks an architect specifies in Phase 4; their internal mechanics for an agent (planning, memory, multi-step loops) are Phase 6 (→ see AEAF Framework §15).

Table 13.2 — Non-determinism design tactics (the contents of the deterministic shell).

TACTIC	FAILURE IT ADDRESSES	HOW IT IS REALISED AT THE BOUNDARY
Bounded outputs & output contract	Free-form output the caller cannot consume.	Constrain the core to a declared schema (structured output / function-call shape); the shell rejects any output that does not validate. The contract, not the model's compliance, is the integration point.
Validation & guardrails at the boundary	Out-of-bounds, unsafe, or policy-violating output passed downstream.	Input guardrails (injection filter, PII screen) before the core; output guardrails (schema, value range, content/safety check, tool-permission and spend veto) after it. Enforced regardless of the core's reasoning.
Fallback & graceful degradation	Core unavailable, over-budget, or repeatedly failing the contract.	Defined alternative: a cheaper/smaller model, a cached prior answer, a deterministic rule, or a reduced-capability safe default. The capability degrades; it does not collapse.

TACTIC	FAILURE IT ADDRESSES	HOW IT IS REALISED AT THE BOUNDARY
Human-handoff path	Low confidence, guardrail veto, or no safe fallback.	A designed escalation to a queue or the accountability owner, carrying full context. Handoff is a first-class success path, not an error (→ see AEAF Framework §19).
Idempotency & compensation	A retried or partially-completed action applied twice, or a wrong action already executed.	Idempotency keys on every world-affecting action so a retry is safe; compensating actions (the saga pattern) to reverse a wrong-but-reversible effect. Irreversible actions are gated to in-the-loop oversight.
Retries aware of cost & variance	Naïve retry that burns budget or just re-rolls the same wrong answer.	Retry only when retrying can help (a transient error, or variance worth re-sampling), within an explicit token/cost and attempt budget; back off; stop and fall back when the budget is spent.
Confidence-aware routing	Treating a low-confidence answer the same as a high-confidence one.	The shell routes on a confidence or quality signal: high → return; medium → cheaper double-check or self-consistency sample; low → fall back or hand off. Confidence is acted on, not displayed.

Two tactics deserve a sentence of emphasis. **Idempotency and compensation** are what make retries and fallbacks safe to use at all: without them, "try again" risks double-charging a customer or sending a duplicate instruction. **Confidence-aware routing** is the tactic that turns a probabilistic core into a tiered system — cheap and fast when the model is sure, careful and human-backed when it is not — and is the application-layer counterpart of the autonomy level set in Phase 6.

2.3 The design rule

The tactics combine into one rule for Phase 4: **for every probabilistic core, name its output contract, its guardrails, its fallback, and its handoff before it ships.** A model call with none of these is not an architecture; it is an unguarded boundary. The shell is the deliverable.

3. Where the agent runtime sits: component versus actor

3.1 The metamodel distinction the application layer must respect

AEAF's metamodel separates three things that classical architecture would have drawn as one box: the **Model** (the reasoning engine), the **agent actor** (the actor that uses the model with tools, memory, an objective, and an autonomy level), and the **Application Component** that hosts it. The rule for this chapter is the one the application layer is responsible for: **an AI service or agent runtime is an Application Component; the agent it hosts is a separate metamodel entity — an agent actor.** The Application Component entity is *extended* in AEAF with a `hostsAgents` relationship for exactly this reason.

Why the distinction is load-bearing, not pedantic:

- **Lifecycle attaches at different levels.** You deploy, scale, patch, and decommission the *runtime* (a component concern). You grant autonomy, set guardrails, and assign an accountability owner to the *agent* (an actor concern, owned in Phase 6). Collapsing them means a deployment decision and a governance decision share one object — and one of them gets lost.
- **The portfolio split follows from it.** The agent runtime is catalogued in the application portfolio (§6); the agent actor it hosts is catalogued in the Agent Catalog and the model it runs in the Model Portfolio (→ see AEAF Framework §15, §21). Same physical deployment, three different governance registers.
- **Many-to-many is normal.** One runtime can host several agent actors; one agent's reasoning can be powered by a model served from a different component (the inference platform, a Phase 5 concern). The component-versus-actor line is what lets the architecture describe this without confusion.

3.2 The three layers in one picture

Figure 13.2 shows the relationship. The **Application Component** (the agent runtime / AI service) is the deterministic shell of §2 made concrete as a deployable unit. It *hosts* the **agent actor** (indigo), which *runs on* a **Model** served by the technology layer. The component is what application architecture owns; the actor is what the Intelligence & Agency domain owns; the model is what the technology and +I domains own jointly.

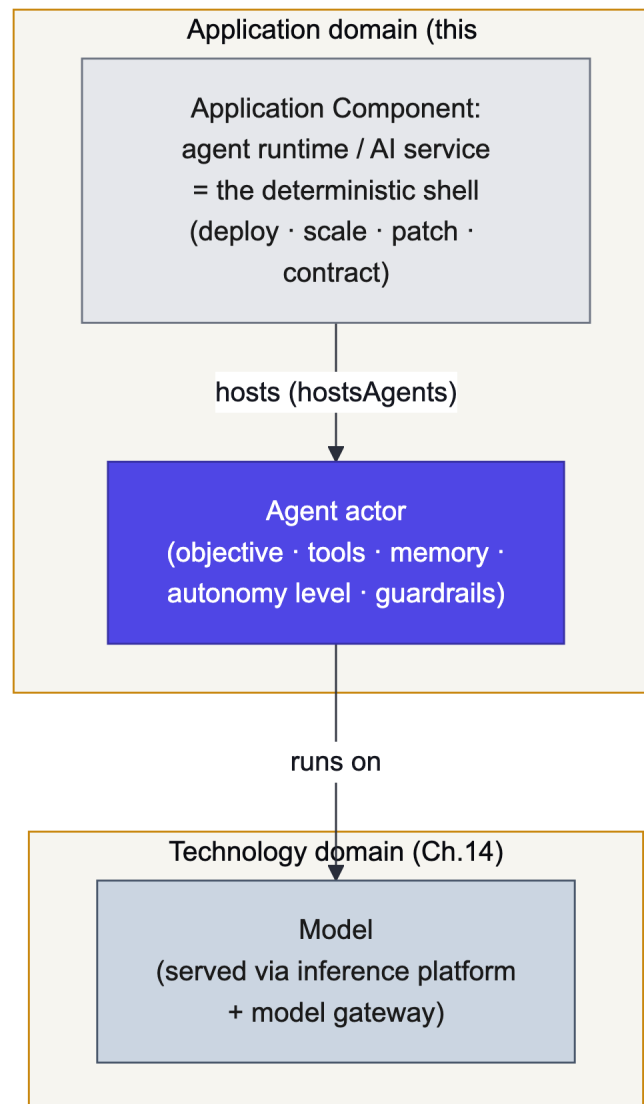


Figure 13.2 — Component hosts actor; actor runs on model. Application architecture owns the runtime component (the shell). The agent actor it hosts is a separate entity governed in the Intelligence & Agency domain. The model the actor runs on is served by the technology layer. The three are deployed together and governed separately. (→ see AEAF Framework §5 for the metamodel, §15 for the agent actor, §14 for the technology layer.)

The practical test: if you find yourself writing "the agent" when you mean "the service we deploy," or "the runtime" when you mean "the actor we hold accountable," the distinction has collapsed and a governance question will fall through the gap.

4. Integration patterns updated for AI

Classical integration patterns survive in the agentic enterprise; they gain AI-specific variants and one new pattern. Table 13.3 maps the pattern, what it integrates, and the non-determinism concern it must now carry. The recurring theme: every pattern that crosses into a probabilistic core does so through the deterministic shell of \$2.

Table 13.3 — Integration patterns updated for the agentic enterprise.

PATTERN	WHAT IT INTEGRATES	AI-SPECIFIC CONCERN IT NOW CARRIES
API gateway	External and internal callers to services.	Unchanged in role; the services behind it may be probabilistic, so the service (not the gateway) carries the shell. Authn/authz/rate-limiting stay here.
Model gateway	Applications to one or more model providers.	New layer (a Phase 5 component): routes by model, enforces token/cost budgets and rate limits, centralises keys, caches prompts, and gives one place for fallback between models. The shell's "fallback to a cheaper model" runs through it.
Tool / function calling	An agent actor to the actions it may perform.	Each tool is an Action in the metamodel with a permission, budget, and reversibility. The shell validates the model's proposed call against the tool schema and the guardrail (permission, spend, reversibility) before execution.

PATTERN	WHAT IT INTEGRATES	AI-SPECIFIC CONCERN IT NOW CARRIES
Model Context Protocol (MCP)	Agents to tools, data sources, and context, via a standard interface.	The emerging standard way to expose enterprise tools/resources to agents without bespoke wiring per agent. Architecturally it makes tools reusable across agents and keeps the permission/guardrail check at the boundary, not in each agent.
Event-driven (async)	Long-running or fan-out agent work to the rest of the estate.	Agent tasks are often slow and variable in duration; queue-and-callback decouples the caller from that latency, enables retry/dead-letter handling, and lets human handoff be an asynchronous step rather than a blocking wait.
Deterministic shell around probabilistic core	The rest of the system to any model-driven component.	The pattern of §2, restated as an integration pattern: no component integrates with a probabilistic core directly; it integrates with the shell's contract. This is the one new mandatory pattern.

Three notes on the table. The **model gateway** is a technology-layer component (→ see AEAF Framework §14) but it is named here because the application's fallback and cost-budget tactics depend on it. **MCP** matters to application architecture because it changes tool integration from per-agent bespoke code to a shared, governed interface — the same architectural move APIs made for system integration. The **event-driven** pattern is what makes the human-handoff path of §2 practical at scale: an escalation becomes a message on a queue, not a thread blocked waiting for a person.

5. Observability of probabilistic components

5.1 Why request-level observability is not enough

Classical observability answers "did the request succeed, how fast, and how often does it fail?" — logs, metrics, traces over requests. For a probabilistic component that is necessary and insufficient, because the failure that matters most (§1.2) is a request that *succeeded* — returned a 200, on time — and was *wrong*. Request-level signals are blind to it. AEAF's rule is: **trace the decision, not just the request**. You must be able to reconstruct, after the fact, *why* a probabilistic component produced the output it did and *which path the shell took*.

5.2 What a decision trace records

A decision trace is the audit object written on every path through the shell in Figure 13.1. It is the evidence base for assurance (→ see AEAF Framework §17) and for incident response (→ see AEAF Framework §20). Table 13.4 contrasts the two layers.

Table 13.4 — Request-level versus decision-level observability.

DIMENSION	REQUEST-LEVEL (CLASSICAL, RETAINED)	DECISION-LEVEL (NEW, REQUIRED FOR PROBABILISTIC COMPONENTS)
Question answered	Did it run, how fast, did it error?	Why this output, with what confidence, down which shell path?
Captured	Latency (incl. time-to-first-token), error rate, throughput, availability, cost.	Inputs and resolved prompt, model + version, retrieved knowledge, tool calls and results, raw output, guardrail verdicts, confidence/quality score, route taken (return / retry / fallback / handoff).
Used for	Performance and reliability operations.	Eval feedback, drift detection, incident reconstruction, the assurance case, accountability.

DIMENSION	REQUEST-LEVEL (CLASSICAL, RETAINED)	DECISION-LEVEL (NEW, REQUIRED FOR PROBABILISTIC COMPONENTS)
Tooling	Metrics + distributed tracing (e.g. OpenTelemetry).	The same tracing spine extended with model/agent spans (retrieval, generation, tool call) and the guardrail/confidence/route attributes above.

The practical implication for Phase 4 design: a probabilistic component's trace fields are part of its contract, specified alongside its output contract. A component whose decisions cannot be reconstructed cannot be assured, governed at runtime, or defended to an auditor. The model and agent spans plug into the same distributed-tracing backbone the rest of the estate uses; this is an extension of existing observability, not a parallel system.

6. The application portfolio in the agentic enterprise

6.1 New classes of application

The application portfolio gains classes that did not exist in a deterministic estate. Each is an Application Component; each, where it hosts an agent, carries the component-versus-actor distinction of §3. Table 13.5 names them.

Table 13.5 — Application classes in the agentic enterprise.

CLASS	WHAT IT IS	RELATIONSHIP TO THE +I PORTFOLIOS
AI service	A bounded model-driven capability exposed behind a contract (classify, extract, summarise, draft). The deterministic shell with no standing objective or autonomy.	Runs a Model (Model Portfolio); usually no agent actor — a single guarded model call, not a goal-pursuing actor.

CLASS	WHAT IT IS	RELATIONSHIP TO THE +I PORTFOLIOS
Copilot	A model-driven assistant inside a human’s workflow; suggests, drafts, answers, with the human deciding and acting.	Typically hosts a low-autonomy agent actor (suggest-only, in-the-loop). Catalogued as both a component and an actor.
Agent runtime	The component that hosts one or more goal-pursuing agent actors, executes their tool calls, and enforces their guardrails.	Hosts agent actors (Agent Catalog); the actors run on models (Model Portfolio) and draw on knowledge corpora (Knowledge-Corpus Catalog).
Orchestration / multi-agent platform	The component that coordinates multiple agents and human handoffs across a workflow.	Hosts/coordinates several agent actors; the orchestration logic itself is Phase 6 (→ see AEAF Framework §15). The platform is the application; the orchestration is the actor design.

6.2 How the application portfolio relates to the model and agent portfolios

The single most important portfolio fact in the agentic enterprise: **the application portfolio, the Model Portfolio, and the Agent Catalog are three registers over the same systems, joined by the hosts/runs-on relationships.** Application architecture (Phase 4) owns the first; the Intelligence & Agency domain (Phase 6) owns the second and third (→ see AEAF Framework §15, §21). They are kept distinct because the lifecycle decisions diverge: you might rationalise an application (retire a redundant copilot) without retiring the model it ran, or swap a model under a stable application without touching the application's contract, or widen an agent's autonomy without changing a line of the runtime that hosts it. Figure 13.3 shows the join.

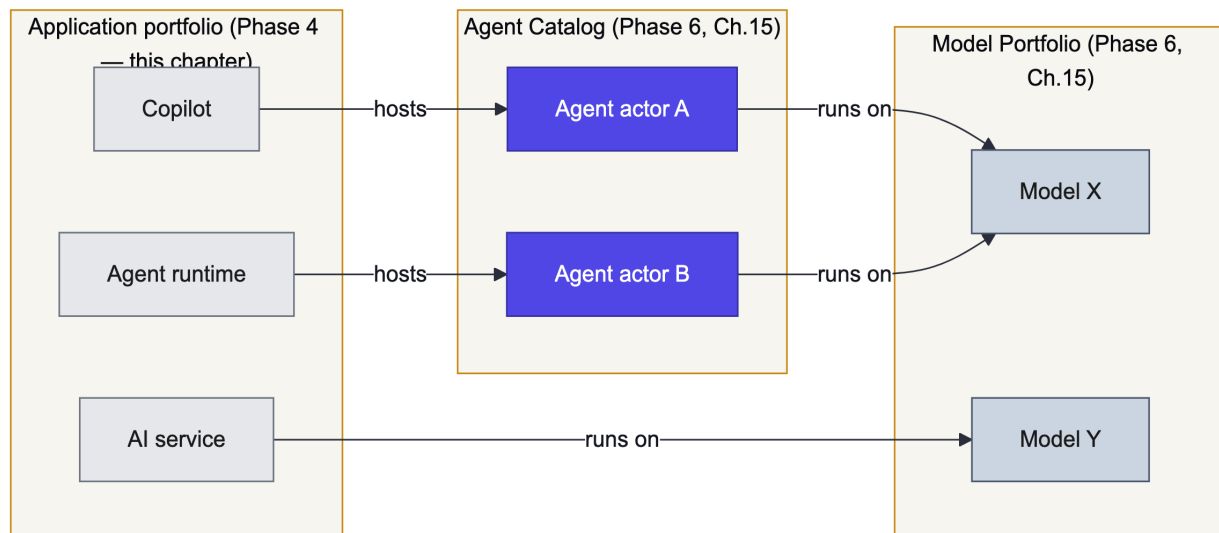


Figure 13.3 — Three registers, one estate. An application component *hosts* agent actors (or, for an AI service, runs a model directly); agent actors *run on* models. The same Model X can power agents hosted by different applications. Rationalisation, model-swap, and autonomy-widening are independent decisions because they act on different registers. (→ see AEAF Framework §15 for the Agent Catalog and Model Portfolio, §21 for the artifact set.)

6.3 Portfolio implication

The classical application-portfolio question — keep, consolidate, refactor, retire — still applies, but it now asks three questions where it asked one: about the *application*, the *agent* it hosts, and the *model* it runs. A portfolio review that treats an agent runtime as a single opaque application will miss that the right move is to keep the runtime, swap the model for a cheaper one, and lower the agent's autonomy — three decisions on three registers, only visible if the registers are kept distinct.

7. What this chapter fixes for the phases around it

Phase 4 hands the rest of the method a portfolio in which every probabilistic component is wrapped in a deterministic shell with a named contract, guardrails, fallback, handoff, and decision-tracing. It does not design the agents inside those shells, the platform beneath them, or the evals that bound them. The boundaries are deliberate:

- **Inward (the core):** the agent actor's objective, tools, memory, planning loop, orchestration, and autonomy level are Phase 6 (→ see AEAF Framework §15). This chapter treats the core as a bounded box and designs its boundary.
- **Downward (the platform):** the inference platform, model gateway, vector stores, and runtime substrate are Phase 5 (→ see AEAF Framework §14). This chapter consumes them; it does not specify them.
- **Centre (assurance):** how evals, thresholds, and assurance cases work — the replacement for one-time testing named in §1.2 and §5 — is the method's centre (→ see AEAF Framework §17). This chapter produces the decision traces that feed it.

The chapter's single claim, restated for the architect who skips to the end: in a non-deterministic world you do not make the component predictable — you make the boundary predictable. The deterministic shell around a probabilistic core (Figure 13.1) is how application architecture stays a discipline when its central assumption no longer holds.

Chapter 14 — Technology architecture

In brief

Technology architecture is the layer beneath the software: the compute, storage, network, middleware, and security infrastructure on which everything else runs. In the agentic enterprise that classical layer does not go away — an agent still needs servers, networks, identity, and key management like any other workload — but a new stratum appears above it that has no counterpart in the deterministic enterprise: **the AI platform**. The AI platform is where inference happens, where a model gateway routes a request to the right model, where a vector store holds the knowledge corpus, where an orchestration runtime carries an agent's reasoning and tool calls, and where the **LLMOps substrate** — eval pipelines, observability, prompt and version management, cost and token controls, caching — keeps probabilistic components measurable and affordable. This chapter is the technology view of that stratum: what it is made of, the integration plumbing that connects it (with **MCP** as the emerging tool-access standard), the cross-cutting concerns that AI amplifies (cost, latency, inference scalability, model data residency, model-supply-chain security), and a Technology Portfolio standards table for the AI stack. It corresponds to **Phase 5 — Technology** of the AEAF Method, the agentic counterpart to TOGAF's ADM Phase D. How a model or an agent is *designed* belongs to Chapter 15; cloud-strategy and FinOps how-to, and the selection playbooks, belong to Book 2.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

14.1 The classical technology layer still applies

Before describing what is new, it is worth being precise about what is unchanged, because the temptation in an AI programme is to treat the AI platform as a self-contained world and forget the ground it stands on.

An agent actor is, at the infrastructure level, an ordinary workload. It runs on compute, reads and writes storage, crosses networks, and must be authenticated, authorized, encrypted, logged, and patched like anything else. The classical technology disciplines — capacity planning, high availability, disaster recovery, network segmentation, identity and access management, secrets management, infrastructure security — all hold without amendment.

CLASSICAL TECHNOLOGY CONCERN	STILL REQUIRED FOR THE AGENTIC ENTERPRISE	WHAT THE AI PLATFORM ADDS ON TOP
Compute & storage	Servers, containers, orchestration, object/block storage.	Accelerated compute (GPU/TPU) where models are self-hosted; high-throughput storage for embeddings and corpora.
Network	Connectivity, segmentation, egress control, latency budgets.	Egress to external inference endpoints becomes a first-class dependency and a data-exfiltration surface.
Identity & access (IAM)	Authenticating and authorizing human and service principals.	Agents need their own machine identities and scoped credentials, distinct from the human who owns them.
Secrets & key management	Storing and rotating credentials, keys, certificates.	Model-provider API keys and tool credentials become high-value, widely-used secrets.
Infrastructure security	Hardening, patching, vulnerability management, monitoring.	Extends to the model supply chain and the orchestration runtime (§14.5).
Observability (infrastructure)	Metrics, logs, traces for hosts and services.	Joined to LLM-level observability so a slow user request can be traced to a slow inference call (§14.3).

The point of the chapter is therefore not that classical technology architecture is superseded. It is that a new stratum sits on top of it, with its own components, standards, and failure modes, and that this stratum is now where most of the architecturally interesting technology decisions are made.

14.2 The AI platform: the new technology stratum

The **AI platform** is the set of technology components that make machine cognition and agency operable: the place a model is invoked, the routing in front of it, the stores that ground it, the runtime that carries an agent's reasoning, and the operational substrate that keeps all of it measured, versioned, and within budget. In the meta-model, every one of these is a **Technology Component**, and the meta-model is explicit that a Technology Component now *hosts* the Model and the Knowledge Corpus rather than treating them as application detail (→ see AEAF Framework §5). The Model and the Agent themselves are designed in the Intelligence & Agency domain (→ see AEAF Framework §15); this chapter is about the platform they run on.

The platform decomposes into seven component groups. Each is a technology choice with its own standards, and each is governed at design time and operated at runtime.

14.2.1 Inference

Inference is the act of running a model to turn an input into an output, and it is the centre of gravity of the whole platform: it is where the cost is incurred, where the latency is dominated, and where the data leaves (or stays inside) the enterprise boundary. The first architecture decision is **where inference runs**, which is a build-versus-buy choice with three working options.

INFERENCE OPTION	WHAT IT IS	CHOOSES IT WHEN	PRINCIPAL TRADE-OFFS
Hosted API (buy)	A model invoked over the provider's public endpoint; the provider owns the weights and the serving infrastructure.	Time-to-value matters; frontier capability is required; volume does not yet justify owning serving.	No infrastructure to run; per-token cost scales with volume; data leaves the enterprise to a third party; capability and price set by the provider.

INFERENCE OPTION	WHAT IT IS	CHOOSES IT WHEN	PRINCIPAL TRADE-OFFS
Self-hosted open-weight (build)	An open-weight model served on the enterprise's own (or its cloud tenant's) accelerated compute.	Data must not leave a boundary; volume is high enough that per-token economics favour owning; specific control or customization is needed.	Full data and version control; fixed cost regardless of token volume; requires accelerated compute, MLOps skill, and capacity planning.
Managed open-weight / dedicated	An open-weight or provider model served on dedicated capacity inside a managed cloud (a hybrid of the two).	A residency or throughput guarantee is needed without operating raw GPU infrastructure.	Residency and capacity guarantees; managed operations; higher fixed cost than shared hosted API.

The build-versus-buy question is rarely answered once for the whole enterprise. The realistic target is a **portfolio**: a frontier hosted model for the hardest reasoning, a cheaper hosted model for routine work, and a self-hosted open-weight model for the data that must not leave — chosen per capability and recorded, with the model-level detail, in the Model Portfolio artifact (⇒ see AEAF Framework §15 and §21).

14.2.2 Model gateway and routing

When more than one model is in use — and in any real deployment more than one model is in use — a **model gateway** sits between the agents and the inference endpoints. It is the AI-platform analogue of an API gateway, and it earns its place by holding the cross-cutting concerns in one governed component instead of scattering them across every agent.

A model gateway typically provides:

- **Routing** — directing a request to the right model by rule (task type, token count, required capability), by classifier, or by cost budget. The **model cascade** — try a cheap model first, escalate to an expensive one only when quality is insufficient — is a routing pattern that materially lowers cost when most requests are easy.¹
- **A uniform interface** — so that an agent calls "the gateway" and the choice of underlying provider becomes a configuration decision rather than code, which is what makes a model portable across providers.

- **Cross-cutting enforcement** — authentication, per-team rate limiting, token-budget quotas, cost attribution, and a single point for caching and logging.
- **Failover** — routing around a degraded or rate-limited provider to a fallback model, so a single provider outage does not stop the enterprise.

The gateway is the natural home for the runtime cost and security guardrails of the platform, because it is the one component every inference call passes through.

14.2.3 Vector store

A **vector store** holds the embeddings that let an agent retrieve from a knowledge corpus by meaning rather than by keyword — the storage technology under retrieval-augmented generation. It is a Technology Component; the corpus it holds is a Data/+I concern designed in the data-and-knowledge architecture (→ see AEAF Framework §12). At the technology layer the architecture decision is which store, against four recurring criteria: scale (vectors held), search style (pure-vector versus hybrid vector-plus-keyword), filtering richness, and whether it should be a managed service or self-hosted, and whether it should be a separate system at all or an extension of an existing database.²

STORE CLASS	REPRESENTATIVE PROPERTY	FITS WHEN
Relational extension	Vector search added to an existing transactional database (e.g. a Postgres extension).	Moderate corpus scale; a relational database is already in place; operational simplicity is valued over peak vector performance.
Dedicated vector database (managed)	A purpose-built store run as a managed service.	Production retrieval at scale; serverless scaling and low operational burden are wanted; some vendor lock-in is acceptable.
Dedicated vector database (self-hosted, open source)	A purpose-built store run on the enterprise's own infrastructure.	Very high scale, advanced filtering, or data-residency control is needed; the team can carry the operational load.
Search-engine / document store with vectors	Vector search inside a full-text search engine or document database.	Hybrid keyword-plus-vector search or a document-centric application is the real requirement.

The same indexing trade-off recurs across all of them — graph-based indexes (HNSW) give high recall and fast search at higher memory cost, inverted-file indexes trade some recall for lower memory, and flat search is exact but does not scale — and the choice belongs in the standards view (§14.6), not in each agent's code.

14.2.4 Orchestration and agent runtime

The **agent runtime** is the execution environment that carries an agent through its loop: receive an intent, reason over a model, choose a tool, call it, observe the result, and continue until the objective is met or a bound is hit. **Orchestration** is the layer above it that coordinates multiple agents, or multiple steps, toward an outcome. In the meta-model this is an Application Component that *hosts* the Agent Actor — the runtime is the technology; the agent is the actor it hosts (→ see AEAF Framework §5).

At the technology layer the runtime must provide, regardless of which framework supplies it: durable state and memory for a long-running agent; the tool-invocation mechanism (with the permission and budget checks that guardrails attach to); retry, timeout, and fallback handling for the non-deterministic calls inside the loop; and the trace emission that makes the loop observable. The design of *what the agent does* inside this runtime — its objective, tools, autonomy level, and guardrails — is the subject of the Intelligence & Agency phase (→ see AEAF Framework §15); the runtime is the platform that executes it.

14.2.5 Prompt and context management

A prompt is not a string buried in code; in a governed platform it is a versioned, testable asset. **Prompt management** is the technology for storing prompt templates with versions and changelogs, testing them against eval datasets before deployment, and rolling them back when a change regresses.³ **Context management** is the adjacent concern of assembling what the model sees on each call — system instructions, retrieved corpus passages, conversation history, tool results — within the model's context window, including the pruning and compression needed to stay inside it. Both are platform capabilities precisely because they must be consistent across agents: a prompt or context defect should be fixed and version-controlled in one place, not re-discovered in each agent.

14.2.6 The LLMOps substrate

The **LLMOps substrate** is the operational layer that keeps probabilistic components measurable, versioned, and affordable. It is the single most important addition the AI platform makes over a classical platform, because a deterministic system can be tested once and trusted, whereas a probabilistic one must be measured continuously. Five capabilities make up the substrate.

LLMOPS CAPABILITY	WHAT IT DOES	WHY THE PLATFORM OWNS IT
Eval pipelines	Run an agent or model against a maintained set of cases and score the output on quality, safety, alignment, and cost dimensions, on a schedule and at every change. ⁴	Evaluation replaces one-time testing for non-deterministic work; it is the technical machinery behind the Eval Suite (→ see AEAF Framework §17) and the runtime assurance loop.
Observability	Capture every request and response with model, prompt version, token counts, cost, latency (including time-to-first-token and P95/P99), cache-hit, and outcome; trace a user request across retrieval and generation spans. ⁵	A probabilistic component's behaviour can only be governed if it is recorded; observability is the evidence base for assurance and for incident response.
Prompt / version management	Version prompts, models, and corpora so that any behaviour can be tied to the exact configuration that produced it.	Reproducibility and rollback are impossible without it; it is also what makes an incident diagnosable.
Cost / token controls	Quotas, budget alerts, rate limits, and cost attribution by team, feature, and capability. ⁶	Token cost is variable and unbounded by default; without controls a single misbehaving agent can run up an unbounded bill.

LLMOPS CAPABILITY	WHAT IT DOES	WHY THE PLATFORM OWNS IT
Caching (incl. prompt caching)	Reuse prior results — semantic caching for near-identical requests, and provider-native prompt caching that reuses a large, stable prompt prefix at a fraction of the input-token cost. ⁷	Caching is the largest single lever on both cost and latency for repeated, prefix-heavy workloads, and it belongs once in the platform rather than in each agent.

These five are not optional add-ons. An agent put into production without an eval pipeline, observability, cost controls, and version management is, in AEAF terms, ungoverned — it has no assurance evidence, no way to detect drift, and no bounded cost. The substrate is the technology that makes the runtime assurance loop (→ see AEAF Framework §20) possible at all.

Reference architecture of the AI platform

Figure 14.1 shows the platform as layers, with the indigo/amber convention: indigo is the machine-intelligence platform, amber marks the points where human governance touches it.

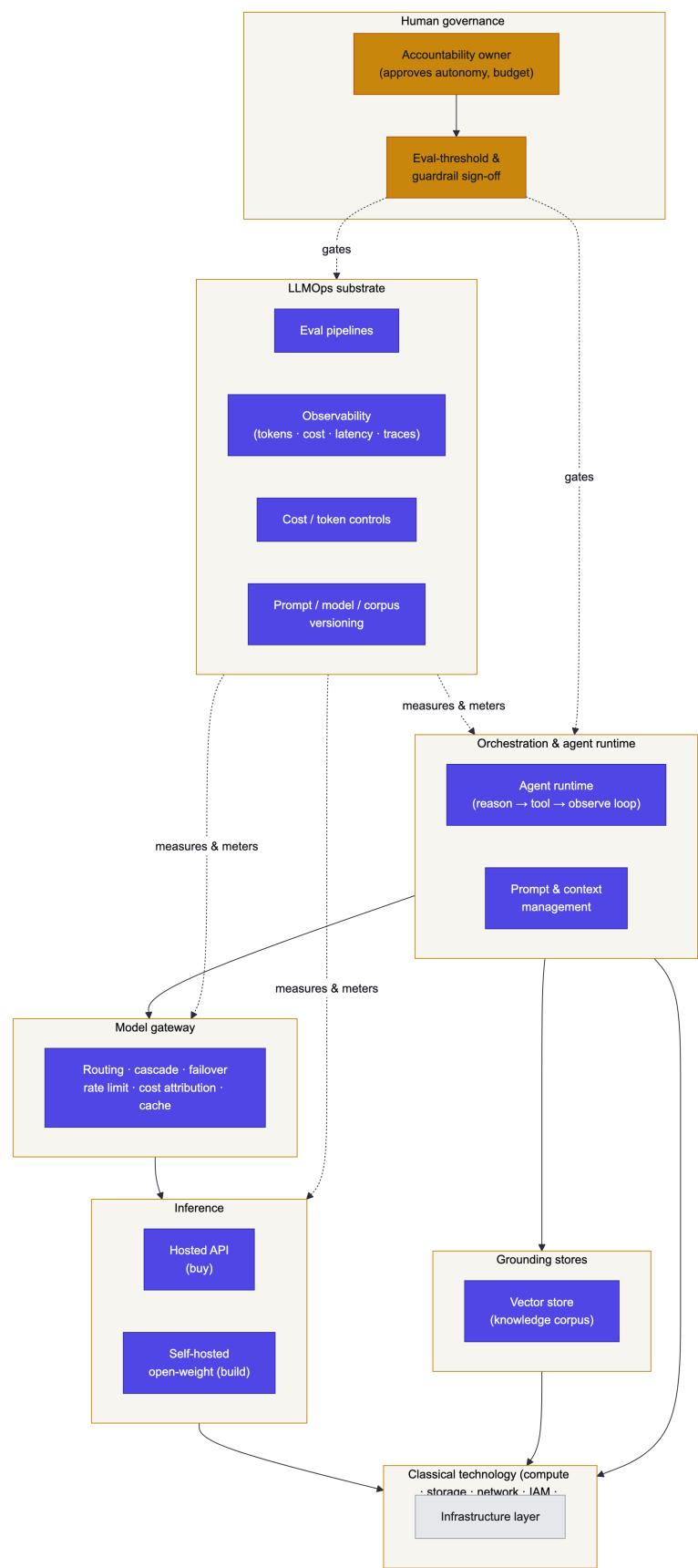


Figure 14.1 — Reference architecture of the AI platform. Indigo components are the machine-intelligence platform; the amber band is where human judgment touches it — the accountability owner who approves autonomy and budget, and the sign-off on eval thresholds and guardrails that gates what may operate. The LLMOps substrate (dotted) measures and meters every layer above the classical infrastructure, which is unchanged from the deterministic enterprise.

14.3 Integration plumbing: MCP and tool gateways as standards

An agent's value comes from acting on the world, and it acts through tools — API calls, queries, transactions, messages. How an agent is connected to those tools is a technology-architecture decision, and the field has been converging on a standard for it.

The **Model Context Protocol (MCP)** is an open protocol for exposing resources (read-only data), tools (actions), and prompts to a model-driven client through a uniform interface, so that a tool integration is built once and reused across agents and applications rather than re-implemented per provider.⁸ Its architectural value over ad-hoc, provider-specific tool wiring is **portability and dynamic discovery**: a tool published as an MCP server is callable by any MCP-capable runtime, and a runtime can discover available tools at runtime instead of having them hard-coded.

INTEGRATION APPROACH	STANDARDIZATION	DISCOVERY	PORTABILITY ACROSS RUNTIMES	FITS WHEN
MCP server / client	Open protocol	Dynamic	High	Multiple tools and agents; reuse across applications wanted; runtime independence valued.
Provider-native function calling	Provider-specific	Static	Low	A single provider is fixed and a one-off integration is enough.

INTEGRATION APPROACH	STANDARDIZATION	DISCOVERY	PORTABILITY ACROSS RUNTIMES	FITS WHEN
Custom per-app integration	None	Static	Low	A single simple source where a standard would be over- engineering.

At enterprise scale a single MCP server per tool is not enough; the platform needs a **tool gateway** (the MCP gateway pattern) in front of the servers, providing centralized authentication, routing, caching, rate limiting, and a single audit point across all tool servers — the same consolidation the model gateway provides for inference, applied to tool access.⁹ A **registry** lets the platform discover which server provides which capability. Two technology rules follow directly from the agentic context: every tool call carries the calling agent's scoped identity (not a shared service account), and every tool call is audited, because a tool call is an agent acting in the world and is the surface where a guardrail's tool-permission and reversibility checks (→ see AEA Framework §5) are enforced.

14.4 Cross-cutting technology concerns amplified by AI

Some technology concerns are not new but are amplified to the point of changing the architecture. Five matter most. The treatment here is at the level of *what the technology architecture must account for*; the how-to of cloud strategy and FinOps is Book 2's (→ see AEA Framework in Practice §14).

CONCERN	WHY AI AMPLIFIES IT	WHAT THE TECHNOLOGY ARCHITECTURE MUST DO
Cost (token economics)	Cost is per-token and unbounded by default; a single verbose agent loop or a runaway multi-agent orchestration can incur cost with no natural ceiling.	Make cost a first-class, metered, attributed quantity: token budgets and quotas at the gateway, the model cascade to keep cheap work on cheap models, and caching (including prompt caching) on prefix-heavy workloads.
Latency	A model call is slow relative to a database query; an agent loop chains several calls plus tool calls, so latency compounds.	Budget latency per interaction class (real-time / interactive / background), exploit streaming (time-to-first-token), cache aggressively, and route latency-sensitive work to faster models.
Scalability of inference	Inference is compute-bound on scarce accelerated hardware; demand is spiky; self-hosted serving must be capacity-planned, hosted endpoints carry rate limits.	Plan inference capacity explicitly — provider rate limits as a real constraint, autoscaling and queueing for self-hosted serving, and gateway failover so a saturated provider does not halt the enterprise.
Data residency / sovereignty for models	Sending data to a hosted model can move regulated or sensitive data across a jurisdictional or organizational boundary at inference time.	Classify what each capability sends to inference, and place inference accordingly — self-hosted or in-region managed serving for data that must not leave, hosted API for the rest. Residency becomes an inference-placement decision, not only a storage decision.

CONCERN	WHY AI AMPLIFIES IT	WHAT THE TECHNOLOGY ARCHITECTURE MUST DO
Security of the model supply chain	The model, its weights, its serving image, and the tools it can call are a new supply chain and a new attack surface (poisoned weights, malicious model artifacts, prompt injection through retrieved content or tool output).	Treat model artifacts and serving images as supply-chain components — provenance, integrity verification, scanning — and treat every input to and output from the model as untrusted (input validation, output filtering, scoped tool permissions).

The model-supply-chain row connects forward to the trust, safety, and accountability architecture (→ see AEAF Framework §18), which treats the full agentic threat surface; here the point is narrower — these are *technology-platform* controls, owned by the platform, not by each agent.

14.5 The technology standards view (AI-stack Technology Portfolio)

A technology architecture commits to standards: for each capability the platform must provide, it states the options considered and the criteria by which one is selected, so that teams build on a governed, coherent set rather than each making its own choice. Table 14.5 is that standards view for the AI stack — a Technology Portfolio extract. It records *capability* → *options* → *selection criteria*; it deliberately does not pick winners, because the right choice is enterprise-specific and the selection *playbook* is Book 2's (→ see AEAF in Practice §14). Vendor names appear in §14.6 only as illustrations of an option class.

AI-STACK CAPABILITY	STANDARD OPTIONS (CLASSES, NOT PRODUCTS)	PRIMARY SELECTION CRITERIA
Inference hosting	Hosted API · self-hosted open-weight · managed/dedicated open-weight	Data residency need · token volume vs fixed-cost crossover · required capability tier · operational capacity

AI-STACK CAPABILITY	STANDARD OPTIONS (CLASSES, NOT PRODUCTS)	PRIMARY SELECTION CRITERIA
Model selection	Frontier model · mid-tier model · small/fast model · domain fine-tuned model	Task complexity · cost sensitivity · latency requirement · context-window need · privacy of input
Model gateway / routing	Build · adopt a gateway product · provider-native routing only	Number of models/providers · need for cascade, failover, quotas, cost attribution in one place
Vector store	Relational extension · managed vector DB · self-hosted vector DB · search-engine with vectors	Corpus scale · hybrid-search need · filtering richness · managed vs self-hosted · residency
Embedding model	Hosted embedding API · self-hosted embedding model	Residency of corpus content · cost at corpus scale · domain fit · dimension/compatibility with store
Agent runtime / orchestration	Framework-based runtime · provider-native agent runtime · build	Multi-agent need · state/memory durability · tool-permission enforcement · observability hooks · portability
Tool integration	MCP server + tool gateway · provider function calling · custom	Tool count · reuse across agents · runtime portability · audit and identity requirements
Prompt management	Git-based · database-backed · hybrid · platform product	Need for analytics/A-B testing · review workflow · scale of prompt estate
Eval & observability	Build on tracing standard · adopt an LLMOps platform · provider-native dashboards	Eval-dimension coverage · trace depth · cost/token attribution granularity · integration with CI gates

AI-STACK CAPABILITY	STANDARD OPTIONS (CLASSES, NOT PRODUCTS)	PRIMARY SELECTION CRITERIA
Caching	Provider-native prompt caching · semantic cache · response cache	Prefix stability and size · request repetition rate · freshness tolerance · cost vs latency target

Each selected standard is recorded against the relevant artifact — inference and model choices in the Model Portfolio, corpus stores in the Knowledge-Corpus Catalog, tool/guardrail enforcement points in the Guardrail/Policy Catalog (→ see AEAF Framework §21) — so the standards view is not a static table but a maintained, traceable commitment.

14.6 A note on vendor surfaces (illustration only)

AEAF is vendor-neutral by design: the framework names *option classes*, and the standards view above is written so it survives any change in the vendor landscape. It is nonetheless useful to anchor the option classes to a concrete surface a reader will recognise, so the following are given strictly as **illustrations of a class, not recommendations**. One major model-provider surface, for example, exposes a hosted inference API with several model tiers (a frontier, a mid, and a fast tier — the model-selection row of Table 14.5), provider-native **prompt caching** that reuses a stable prompt prefix at a large discount (the caching row),⁷ and an agent runtime / SDK that supplies the reason-tool-observe loop and tool wiring (the orchestration row); the same models are also offered through several hyperscaler platforms for enterprises that need the inference governed inside their existing cloud tenant and residency boundary (the managed/dedicated inference row). Equivalent surfaces exist from other providers, and open-weight models fill the self-hosted column. The architecture decision is which *class* fits a capability; which vendor realises the class is a procurement and Book 2 selection decision, made against the criteria in Table 14.5 and re-made as the market moves.

14.7 Phase 5 inputs and outputs

Phase 5 — Technology takes the application and knowledge architectures and produces the AI-platform technology architecture and its standards, handing them forward to the Intelligence & Agency phase that designs the models and agents the platform will host. Table 14.7 states the phase boundary.

	ITEM	FROM / TO
Inputs	Application architecture: the agents-as-actors, services, integration, and non-deterministic-design decisions to be hosted.	← Phase 4 (→ see AEAF Framework §13)
	Data & knowledge architecture: the knowledge corpora and embeddings to be stored and served.	← Phase 3 (→ see AEAF Framework §12)
	Capability allocations and target autonomy levels: which capabilities run on agents, at what oversight.	← Phase 2 (→ see AEAF Framework §11)
	Architecture principles (incl. runtime-enforced) and the risk/consequence envelope.	← Preliminary & Vision (→ see AEAF Framework §8, §10)
	Non-functional intents: latency, cost, residency, and availability bounds.	← Intent & Assurance (centre) (→ see AEAF Framework §17)
Outputs	The AI-platform reference architecture: inference, gateway, vector store, runtime, prompt/context management, LLMOps substrate, on the classical technology layer.	→ Phase 6 (→ see AEAF Framework §15)
	The AI-stack Technology Portfolio / standards view (Table 14.5).	→ Phase 6, Phase 7 (→ see AEAF Framework §16)

ITEM	FROM / TO
The integration-plumbing standard: MCP/tool-gateway pattern, identity and audit rules for tool access.	→ Phase 6
Cross-cutting technology positions: cost-control, latency-budget, inference-scaling, residency-placement, and supply-chain-security stances.	→ Phase 6, Trust & Safety (→ see AEAF Framework §18)
Platform contributions to the Model Portfolio, Knowledge-Corpus Catalog, and Guardrail/Policy Catalog enforcement points.	→ Phase 9, Phase 21 (→ see AEAF Framework §21)
Technology constraints and gaps feeding migration planning.	→ Phase 8 (→ see AEAF Framework §16)

The handoff to Phase 6 is the pivot of the method: the technology architecture says *what platform exists and on what standards*, and the Intelligence & Agency phase designs *what cognition and agency will run on it*. The two are iterated — a model or autonomy decision in Phase 6 routinely sends a new requirement back to the platform (a residency constraint, a latency budget, a tool-gateway capability), which is the normal concurrent iteration of the AEAF Method, not a defect.

14.8 The Technology Radar

The AI stack turns over faster than any architecture document. A protocol that looks essential this year can be superseded the next, and the risk is not that the new thing is bad — it is that an emerging technology quietly becomes **load-bearing by default**, adopted into the platform before anyone decided it should be. The Technology Radar is the lightweight discipline that prevents this: a periodic, deliberate placement of each emerging or connecting technology into one of four rings, so adoption is a recorded decision rather than a drift.

The four rings are the established radar convention, read as a maturity-of-commitment scale:

RING	MEANING	WHAT IT PERMITS
Adopt	Proven for our context; the default choice for new work.	Build on it as a standard (the Technology Portfolio, §14.5).
Trial	Promising; worth a bounded pilot on a real but non-critical capability.	Use behind a guardrail, with an exit plan, on contained work.
Assess	Worth understanding and watching; not yet for production.	Research and prototype only; no production dependency.
Hold	Do not start new work on it — declining, risky, or superseded.	Stop new adoption; plan migration off existing use.

The radar is reviewed on a cadence (a quarter is typical for a fast-moving stack) and each placement is a justified, dated decision with a named owner. A technology may only move into **Adopt** — i.e. become a platform standard — by an explicit decision, never by accumulated usage. This is the mechanism behind the principle-led, protocol-pluggable principle (→ see AEAF Framework §8, R5): the framework commits to capabilities and option-classes, and the radar is where a specific connecting technology earns, or loses, its place.

Three current examples show how connecting technologies are placed without becoming dependencies. Each is a *capability an agent reaches through a guardrailed action*, governed by an autonomy level and a named accountability owner — not a platform the architecture is built on.

TECHNOLOGY	WHAT IT IS	RADAR PLACEMENT (ILLUSTRATIVE)	WHY IT IS AN ACTION, NOT A DEPENDENCY
x402	Agentic payments over the HTTP 402 Payment Required status — an agent settles a micro-charge inline to access a paid resource or API.	Assess	The agent reaches it through a bounded, spend-limited action under a guardrail (per-call and per-session budgets, S5; trust-boundary rules, S4), at a set autonomy level with a named accountability owner. The platform exposes “pay for a resource” as a governed tool, not as a payment rail the architecture depends on.
Agent-to-agent (A2A) protocols	Open protocols for one agent to discover, delegate to, and exchange results with another agent across an organizational boundary.	Assess / Trial	A2A is an integration option-class alongside MCP and provider-native calling (§14.3); a cross-boundary delegation is an action carrying the calling agent’s scoped identity and an audit record, bounded by guardrails — committed to as a capability (“delegate to an external agent”), not to one protocol.

TECHNOLOGY	WHAT IT IS	RADAR PLACEMENT (ILLUSTRATIVE)	WHY IT IS AN ACTION, NOT A DEPENDENCY
MCP successors / competitors	Protocols that may extend, compete with, or replace MCP for tool and resource access.	Watch under the portability principle	AEAF commits to the portability and dynamic-discovery property (§14.3), not to MCP specifically. A successor that better serves that property moves toward Trial/ Adopt on its merits; MCP moves toward Hold if it is superseded. The principle outlives the protocol.

The radar reinforces vendor and technology independence directly: nothing reaches **Adopt** without a recorded decision against selection criteria, which is the design-time face of the technology & vendor independence principle (→ see AEAF Framework §8, R4). A named vendor or protocol is a realisation of an option-class, placed on the radar and re-placed as the market moves — never a commitment baked into the standard.

The Technology Radar is a **supporting governance view, not one of the nine core AEAF artifacts** (→ see AEAF Framework §21). It is maintained alongside the Technology Portfolio standards view (§14.5) as a living, dated decision log for emerging and connecting technologies; the canonical artifact set remains the nine of Chapter 21.

1. Model-cascade and routing patterns (rule-based, classifier-based, cost-aware), and the cost table behind tier selection, follow the practice summarised in *LLMOps Best Practices* (LLMOPS-KB-001, v1.0, 2025-10-02), §Model Selection & Optimization. [↗](#)
2. Vector-store classes, indexing trade-offs (HNSW / IVF / flat), and the selection criteria summarised here follow *Vector Database Comparison* (VECTOR-DB-COMP-001, v1.0, 2025-10-02). [↗](#)
3. Prompt-management approaches (Git-based, database-backed, hybrid), versioning, and eval-dataset testing follow *LLMOps Best Practices* (LLMOPS-KB-001), §Prompt Management. [↗](#)

4. Eval-dataset and scoring practice follows *LLMOps Best Practices* (LLMOPS-KB-001), §Prompt Testing / §Continuous Evaluation. [↪](#)
5. Observability metrics (TTFT, total latency, P50/P95/P99, cost-per-request, cache-hit, distributed tracing across retrieval and generation spans) follow *LLMOps Best Practices* (LLMOPS-KB-001), §LLM Observability & Monitoring. [↪](#)
6. Quotas, budget alerts, rate limiting, and cost attribution follow *LLMOps Best Practices* (LLMOPS-KB-001), §Token & Cost Management. [↪](#)
7. Provider-native prompt caching reusing a stable prompt prefix at a large input-token discount, and semantic caching, follow *LLMOps Best Practices* (LLMOPS-KB-001), §Caching Strategies. The vendor mention in §14.6 is illustrative of the option class, not a recommendation. [↪↪](#)
8. MCP fundamentals (resources, tools, prompts; client/server/transport; standardization, dynamic discovery, and portability over provider-specific function calling) follow *Model Context Protocol (MCP)* (MCP-KB-001, v1.0, 2025-10-02), §MCP Architecture Overview. [↪](#)
9. The MCP gateway and registry (service-discovery) patterns and their centralized auth/routing/caching/rate-limiting/audit benefits follow *Model Context Protocol (MCP)* (MCP-KB-001), §Enterprise MCP Integration. [↪](#)

Chapter 15 — Intelligence & Agency architecture

In brief. Intelligence & Agency is the fifth AEAf architecture domain — the +I in BDAT+I — and Phase 6 is where its content is designed. The four classical domains describe the enterprise's business, data, applications, and technology; none of them has a home for the population of models and agents that now perform work, the knowledge they draw on, the guardrails that bound them, the evals that assure them, and the autonomy they are granted. This chapter designs that population as architecture, not as code. It defines what Phase 6 produces; gives the **anatomy of an agent** (model, knowledge, memory, tools, objective, guardrails, oversight) and the single-agent reasoning patterns at architecture level; treats **model selection** as a first-class architecture decision and the **Model vs Agent** distinction in practice; shows how an agent draws on the cognitive substrate from Chapter 12; sets out **orchestration** from single-agent to multi-agent, and when multi-agent is justified rather than over-engineering; treats **tools and actions** under least-privilege with MCP as the integration standard; designs **guardrails and autonomy** as defense in depth across input, reasoning, output, and action; and requires that every agent is built with its **eval suite**. It closes with the Phase 6 inputs and outputs, including the four artifacts this domain owns.

TOGAF® is a registered trademark of The Open Group. AEAf is an independent framework and is neither derived from nor endorsed by The Open Group. Where TOGAF's Phases B–D produce the Business, Data, Application, and Technology architectures, AEAf adds Phase 6 — Intelligence & Agency — for a population of actors TOGAF's content metamodel was never built to describe.

15.1 What Phase 6 produces: the designed agentic workforce

Phase 6 designs the **agentic workforce itself** — the actual population of models and agents the enterprise will run, how they are orchestrated, what knowledge they use, what they are permitted to do, how they are bounded, and how their behaviour is assured. The four classical domains lead up to this point: Business & Capability (Phase 2) decided *which*

capabilities are allocated to agents and at what autonomy; Data & Knowledge (Phase 3) built the **cognitive substrate** agents reason over; Application (Phase 4) designed components to behave correctly around non-deterministic parts; Technology (Phase 5) stood up the **AI platform** — inference, gateways, vector stores, the agent runtime. Phase 6 is where those inputs become a designed set of actors.

The distinction that organises the whole phase is the one the meta-model draws and TOGAF would have collapsed: **Model, Agent, and Application Component are three things, not one** (→ see AEA Framework §5).

- A **model** is the capability to reason and generate. It is passive — given an input it returns an output. It does not act.
- An **agent** is an *actor*: a model placed in a loop with an objective, memory, knowledge access, tools, guardrails, and a granted autonomy level. It perceives, decides, and acts toward a goal with limited human direction.
- An **application component** is the runtime that hosts the agent — a box in the Application domain. The agent it hosts is modelled separately, in +I.

This is the central modelling correction of the framework (→ see AEA Framework §3). Treating an agent as "an application with an LLM inside" buries autonomy, guardrails, accountability, and evaluation inside an application box where they cannot be governed. Phase 6 lifts them out and designs them explicitly.

What the phase produces is captured in four artifacts it owns, defined in Chapter 21 and templated with worked examples in Book 2 (→ see AEA Framework §15):

PHASE 6 ARTIFACT	WHAT IT RECORDS	ONE-LINE PURPOSE
Agent Catalog	Every agent actor: purpose, model, knowledge corpora, tools, autonomy level, guardrails, eval suite, accountability owner.	The register of the non-human workforce.
Model Portfolio	The models in use: provider, version, modality, context, cost/latency profile, risk profile, and which agents each powers.	The model layer as a managed portfolio, not ad-hoc choices.

PHASE 6 ARTIFACT	WHAT IT RECORDS	ONE-LINE PURPOSE
Guardrail/Policy Catalog	Machine-enforceable policies and the guardrails that enforce them, mapped to enforcement point and fail action.	What each agent is actually constrained by.
Autonomy-Level Classification	The autonomy scale and each agent capability's assigned level, with required controls per level.	How much independent action each agent is granted, and on what conditions.

A fifth artifact, the **Eval-Suite Specification**, is co-owned with Chapter 17 (Intent & Assurance): Phase 6 builds the eval suite *with* the agent; Chapter 17 owns the assurance framework that consumes it. Two further +I artifacts — the **Knowledge-Corpus Catalog** (Chapter 12) and the **Trust & Accountability Matrix** (Chapters 18–19) — are referenced by Phase 6 but produced elsewhere.

The rest of this chapter is the design content behind these artifacts. Three concerns are deliberately deferred: the **assurance framework** that turns eval results into an assurance case is Chapter 17; **runtime governance** — enforcement, drift, incident response while the agent operates — is Chapter 20; and the **principle catalog** that guardrails compile from is Chapter 8. Build playbooks and the artifact templates are Book 2 (→ see AEAf in Practice §10, §12, §15).

15.2 The anatomy of an agent

An **AI agent** in AEAf is a software actor that pursues an objective by reasoning over a model, drawing on knowledge and memory, taking actions through tools, all within guardrails, at a granted autonomy level, answerable through a named accountability owner. Six parts make it an actor rather than a function. Designing an agent means designing all six; leaving any of them implicit is where agentic systems fail.

PART	WHAT IT IS	ARCHITECTURE DECISION IT FORCES
Objective	The bounded intent the agent pursues — outcome, bounds, priority, acceptance signal. Not a list of steps.	What is the agent for, and what must it never do in pursuit of it? (→ see AEAf Framework §17 for Intent.)

PART	WHAT IT IS	ARCHITECTURE DECISION IT FORCES
Model	The reasoning engine (§15.3). One model may power many agents; one agent may route across several.	Which model, at what capability/cost/latency, with how much control?
Knowledge & memory	Grounding from the cognitive substrate (RAG corpora, graphs) plus the agent's own memory (§15.4).	What may the agent know, from where, how fresh, and what does it remember across turns and sessions?
Tools / actions	The agent's ability to affect the world — API calls, transactions, messages (§15.6).	What may the agent do, with what permission, budget, and reversibility?
Guardrails	Constraints enforced at runtime regardless of the agent's reasoning (§15.7).	What bounds the agent when its reasoning is wrong?
Oversight & accountability	The human relationship to the agent's actions — mode, escalation, and the named owner answerable for outcomes (§15.5).	Who watches, when does the agent hand back, and who answers for what it does?

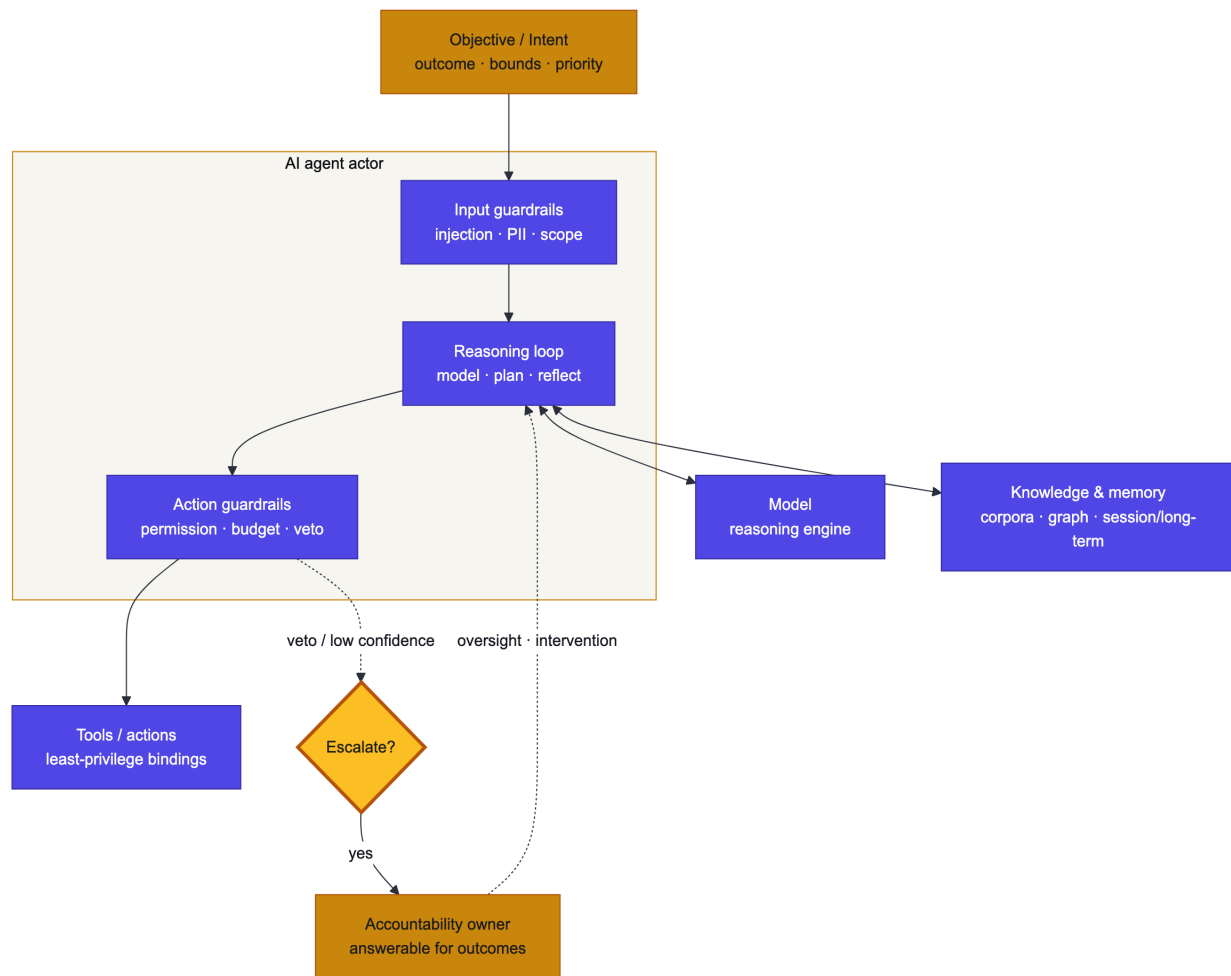


Figure 15.1 — The anatomy of an agent. Indigo is machine intelligence: the model, the knowledge and memory it draws on, the reasoning loop, the tools it acts through, and the input/action guardrails that bound it. Amber is human judgment: the objective that sets what good looks like and the accountability owner who oversees on-the-loop and receives escalations. The agent is the whole assembly — the model is only its engine.

Single-agent reasoning patterns, at architecture level

How the agent's reasoning loop is shaped is an architecture choice, not just an implementation detail, because each pattern changes the agent's latency, cost, failure modes, and what must be observed and bounded. AEAF treats the loop shape as a documented property of the agent, recorded in the Agent Catalog. Four patterns recur.^{[1](#)}

PATTERN	WHAT THE LOOP DOES	USE IT WHEN	ARCHITECTURE COST / RISK
Tool-use	Single decision: the model selects a tool, the tool runs, the result returns.	The task is one bounded action behind a model decision (look up, calculate, fetch).	Lowest cost; risk is a wrong tool call — bound by tool permissions.
ReAct (reason + act)	The agent alternates thought → action → observation over several steps until it has enough to answer.	Multi-step tasks needing several tools and visible intermediate reasoning.	More calls, higher latency; the reasoning trace is a monitoring surface and an injection surface.
Plan-then-execute	The agent first produces a plan, then executes the steps, re-planning on failure.	Tasks with dependencies where a plan can be reviewed before acting.	The plan is an oversight artifact — it can be gated before execution; re-planning loops need a step budget.
Reflection	The agent critiques and revises its own output before returning it.	Quality-sensitive outputs where a second pass measurably reduces errors.	Roughly doubles token cost; the critic can be a separate, smaller model.

These are not exclusive. A production agent commonly plans, uses ReAct within steps, and reflects on the result. The architecture point is to **name the pattern, budget its steps, and decide what is observed** — an unbounded loop is the most common way an agent runs up cost or drifts off-objective.

15.3 The model layer and model selection as an architecture decision

The **model** is the agent's reasoning engine and nothing more — a passive function that maps input to output. Choosing it is one of the highest-leverage decisions in Phase 6, and AEAF treats it as an architecture decision with recorded trade-offs, not a default to the largest available model. The model layer as a whole is managed as the **Model Portfolio** — the TIME-equivalent lifecycle for models (→ see AEAF in Practice §11).

Six criteria drive the choice. They trade off against each other; the art is fitting the model to the *capability*, not to the catalogue.

CRITERION	THE QUESTION	WHY IT IS AN ARCHITECTURE DECISION
Capability	Can the model do the reasoning the capability needs, reliably enough?	Under-capable models fail the eval suite; over-capable models waste cost on simple work.
Cost	What does each call cost at expected volume?	At agent volumes (thousands of calls/day) model cost dominates run cost; a 5× price gap is an architecture-level number.
Latency	How fast must the agent respond?	An in-the-loop human flow needs sub-second response a slow model cannot give; a batch agent does not care.
Context window	How much must the model see at once — long documents, long histories, large tool outputs?	Sets the ceiling on knowledge and memory the agent can hold per call (\$15.4); a hard architectural limit.
Control & deployment	Can it run where the data must stay (region, VPC, on-prem)? Can it be versioned and pinned?	Drives data residency, supply-chain trust, and reproducibility — a governance constraint, not a preference. ²
Modality	Text only, or vision / audio / structured output?	A capability needing document images or speech rules out text-only models.

No single model wins all six. The architecture response is a **multi-model strategy**: place each model where its profile fits, behind a model gateway so agents are not bound to one provider.

MULTI-MODEL PATTERN	HOW IT WORKS	WHEN TO USE IT
Cascade	Try a cheap, fast model first; escalate to a stronger model only when the cheap one is low-confidence or fails a check.	High volume with a long tail of hard cases; most traffic is easy.
Router	A lightweight classifier sends each request to the model best suited to it.	Mixed workloads where request type predicts the right model.
Specialist	A capability binds permanently to a model chosen for it (e.g. a code model for code, a long-context model for document work).	The capability's needs are stable and distinctive.
Ensemble / judge	Several models answer; a judge model or vote selects or merges.	High-stakes outputs where redundancy justifies the cost.

Recording the choice matters as much as making it. Each agent in the Agent Catalog names its model; each model in the Model Portfolio names its profile and the agents it powers — so when a model is deprecated or a cheaper one appears, the blast radius is visible and the change is governable rather than a silent swap.

15.4 How an agent draws on the cognitive substrate

The knowledge an agent reasons over is built in Phase 3 and catalogued as the **Knowledge-Corpus Catalog** (⇒ see AEAF Framework §12 for the cognitive substrate — corpora, embeddings, vector indexes, memory stores, knowledge graphs). Phase 6's concern is the **agent side**: how the agent *uses* that substrate, and what that costs in latency, accuracy, and exposure. Four forms of knowledge use recur, and most production agents combine them.

KNOWLEDGE USE	WHAT THE AGENT DOES	PHASE 6 DESIGN CONCERN
Retrieval (RAG)	Fetches relevant passages from a corpus and grounds its answer in them.	Which corpora may this agent read, at what classification? Retrieved content is an indirect injection surface — it is bounded by input guardrails (§15.7). ³
Graph / structured lookup	Queries a knowledge graph or structured store for facts and relationships.	Precise, auditable facts vs. the fuzziness of retrieval; the access policy on the structured source.
Session memory	Remembers earlier turns within a task or conversation.	What is kept in-context vs. summarised; the context-window budget (§15.3) sets the ceiling.
Long-term memory	Persists learnings, preferences, or state across sessions.	Governance of what an agent is allowed to remember — long-term memory is personal data and a drift source; it has a classification and a retention rule.

Two design rules follow, both with safety weight. First, **what an agent may retrieve is a permission, not a capability** — an agent reads only the corpora its accountability owner has granted, at its clearance, recorded in the catalog. Second, **memory is governed**: long-term memory that accumulates without a retention and classification rule is how an agent silently drifts, leaks, or holds personal data it should not. The agent side of knowledge is therefore as much an access-and-governance decision as a retrieval-quality one.

15.5 Orchestration: single-agent, multi-agent, and human-in-the-loop

Orchestration is how agents and humans are arranged to perform a capability together. The first orchestration decision is the most consequential and the most often gotten wrong: **single-agent or multi-agent**. AEAF's default is single-agent. Multi-agent systems are justified by genuine structure in the problem, not by the appeal of the pattern.

A single agent with a well-chosen model, good tools, and the right reasoning pattern handles most capabilities. Reach for multiple agents only when one of these holds:

- **Genuinely distinct specialisms** — sub-tasks need different models, knowledge, or tool permissions that one agent cannot cleanly hold (e.g. a research agent that may read broadly but not write, and a writing agent with no external read access).
- **Separation for control** — splitting agents so a high-privilege action sits behind a different, more-tightly-bounded agent than the one that proposes it (read/write separation as an architecture, not just a tool flag).
- **Parallelism that pays** — independent sub-tasks that genuinely run concurrently and whose coordination cost is less than the time saved.

When none of these holds, multi-agent is **over-engineering**: it multiplies cost (every agent is model calls), multiplies failure modes (agents miscommunicate, loop, or amplify each other's errors), and multiplies the surface to govern. A second agent must earn its place the way a second hire would.

ORCHESTRATION PATTERN	STRUCTURE	WHEN IT FITS	MAIN RISK
Single-agent	One agent, one loop, its own tools and knowledge.	The default; most capabilities.	Loop runs unbounded — needs step/budget caps.
Supervisor / worker	A supervisor decomposes the task and delegates to specialist workers, then synthesises.	Distinct specialisms or tool/permission separation.	Supervisor becomes a bottleneck or mis-delegates; each worker needs its own bounds.
Collaborative (peer)	Several agents work a shared task over shared context, reaching a result by consensus or merge.	Tasks benefiting from diverse approaches or cross-check.	Coordination overhead; agents amplifying each other's errors.
Pipeline (sequential)	Output of one agent is input to the next, in a fixed chain.	A task with clean, ordered stages.	A failure mid-chain propagates; needs handoff validation between stages.

Human-in-the-loop is an orchestration step, not an afterthought. Where a human approves, monitors, or receives an escalation, that human is a node in the orchestration with a defined role, trigger, and intervention right — placed where consequence and

reversibility demand it (the oversight modes of Chapter 19). Figure 15.2 shows a supervisor/worker orchestration with the human placed as an on-the-loop reviewer and the escalation target.

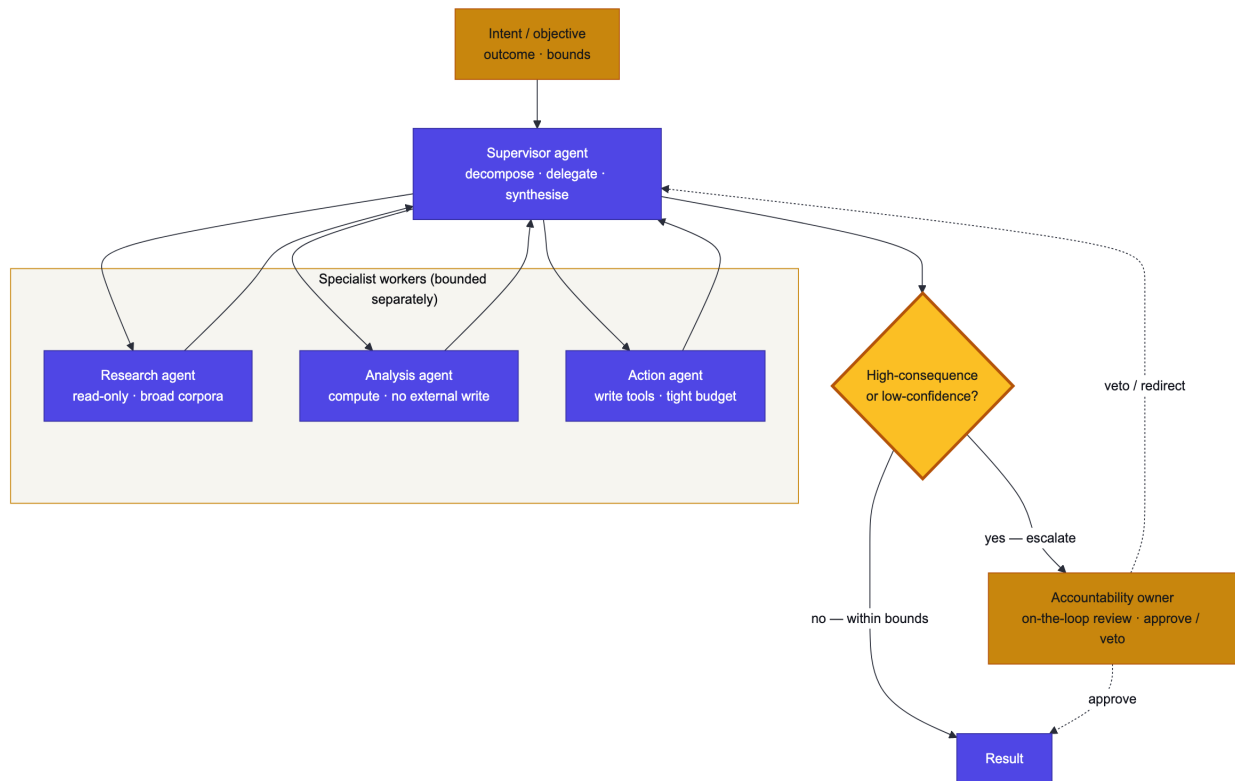


Figure 15.2 — Multi-agent orchestration with human oversight. A supervisor agent (indigo) delegates to specialist workers that are bounded *separately* — the research worker reads broadly but cannot write; the action worker can write but on a tight budget. The human accountability owner (amber) is an on-the-loop reviewer placed at the escalation gate: routine, in-bounds results pass through; high-consequence or low-confidence ones route to the human, who can approve, veto, or redirect. The human is a designed node in the orchestration, not a layer bolted on top.

15.6 Tools and actions: the agent's ability to affect the world

A model that only generates text is harmless; an agent becomes consequential the moment it can **act**. In AEAF an **Action / Tool** is a first-class entity: a concrete capability to affect the world that an agent may invoke — an API call, a transaction, a message — carried with its permission, budget, reversibility, and audit (→ see AEAF Framework §5). Designing an agent's tools is designing the only way it touches reality, and it is governed accordingly.

Five properties are designed for every tool an agent holds:

PROPERTY	WHAT IT SETS	DEFAULT IN AEAF
Permission	Which agents may call this tool, on what data.	Least privilege — default deny; each tool granted explicitly to each agent.
Read vs write	Whether the tool only observes or also changes the world.	Read/write separation — agents default to read-only; write tools require explicit elevation.
Budget	Per-session limits: max calls, max writes, max spend.	Every action-taking agent has an action budget; an unbounded agent has none of these.
Reversibility	Whether the action can be undone, and at what cost.	Irreversible or high-consequence actions are gated to a human (in-the-loop) regardless of autonomy level.
Audit	The record of every invocation: who, what, when, result.	Every tool call is logged; this is the runtime evidence trail (→ see AEAF Framework §20).

These properties are why the meta-model separates the Action entity from the agent: an agent's *reach* — what it can do to the world — is governed independently of its *reasoning*. An agent's reasoning may be wrong; its permissions and budgets bound the damage that a wrong decision can do.

MCP is AEAf's recommended tool-integration standard. The Model Context Protocol is an open protocol for exposing tools (actions an agent may invoke) and resources (data an agent may read) to agents through a uniform interface, with dynamic discovery, rather than provider-specific function-calling wired per application.⁴ Three properties make it the architecture default for the tool layer:

- **Portability** — a tool exposed as an MCP server is callable by any MCP-aware agent, decoupling the tool from the model and the agent from the provider. This is the same portability AEAf requires of principles and capabilities: not bound to one vendor's surface.
- **A governance choke point** — an enterprise **MCP gateway** centralises authentication, authorisation, rate limiting, and audit for tool access across all agents, so least-privilege and budgets are enforced in one place rather than re-implemented per agent.
- **Resource/tool distinction** — MCP separates read-only *resources* from action-taking *tools*, which maps directly onto AEAf's read/write separation: an agent can be granted resources broadly and tools narrowly.

The architecture decision is to expose enterprise actions and data as MCP servers behind a governed gateway, and to bind agents to them through least-privilege permissions recorded in the Agent Catalog — not to hard-wire tools into each agent.

15.7 Guardrails and autonomy by design

An agent's reasoning is probabilistic and will sometimes be wrong. **Guardrails** bound its behaviour at runtime *regardless of that reasoning* — they enforce even when the agent is mistaken, manipulated, or pursuing the objective in a way its designer never intended (→ see AEAf Framework §8 for how a runtime principle compiles into a guardrail). Phase 6 designs guardrails as **defense in depth** across four layers, so a failure at one layer is caught by the next; no single control is trusted alone.⁵

GUARDRAIL LAYER	WHERE IT SITS	WHAT IT CATCHES	EXAMPLE
Input	Before the model reasons.	Prompt injection (direct and via retrieved content), out-of-scope requests, PII that should not enter the model, oversized inputs.	Reject a retrieved document that contains an instruction to ignore policy.
Reasoning	On the agent's intermediate steps.	Goal drift, scope creep in a plan, deceptive or inconsistent chain-of-thought.	Flag a plan that adds a step outside the granted objective.
Output	Before a response leaves the agent.	Unsafe, ungrounded, or policy-violating content; leaked data; malformed structured output.	Block an answer not grounded in the cited sources.
Action	Before a tool executes.	Out-of-permission calls, over-budget actions, irreversible actions without approval, egress of sensitive data.	Veto a payment over the agent's spend limit and escalate.

The binding design rule is **fail-closed**: when a guardrail cannot determine that an action is safe, it denies and escalates rather than permits. An agent that fails open — acting when its bound is uncertain — defeats the purpose of having the bound. Every guardrail therefore declares its **fail action** in the Guardrail/Policy Catalog: deny, escalate, or constrain.

Autonomy is designed within a graded scale, not granted wholesale. An agent's **autonomy level** is how much independent action it is granted for a capability, and it is the single dial that sets which guardrails and oversight mode are required. AEAf uses a five-level classification, L0–L4 (the canonical definitions are in Chapter 4; the operating model that applies them is Chapter 19; the recorded result is the Autonomy-Level Classification artifact). At Phase 6 the architect *assigns a level per agent capability and provisions the controls that level requires*.

LEVEL	WHAT THE AGENT DOES	OVERSIGHT MODE	CONTROLS REQUIRED IN DESIGN
L0 — Assistive	Suggests; a human takes every action.	In-the-loop (human acts)	Output guardrails; logging.
L1 — Bounded action, confirm	Proposes an action; a human confirms before it executes.	In-the-loop (human approves)	Input/output guardrails; approval gate on every action.
L2 — Bounded action, on-the-loop	Acts within tight bounds; a human monitors and can intervene.	On-the-loop	All four guardrail layers; action budget; intervention/kill path.
L3 — Autonomous within bounds	Acts independently within guardrails; human reviews after the fact and on exception.	On/Out-of-the-loop	L2 controls + per-action audit; drift monitoring; defined escalation.
L4 — Autonomous, broad bounds	Acts independently over wider scope and longer horizons.	Out-of-the-loop	L3 controls + mandatory pre-deployment red-team; interpretability/anomaly monitoring; strong runtime loop.

Two rules govern the assignment. First, **the level is set by consequence and reversibility, not by the model's capability** — a capable model performing a high-consequence, irreversible action is held at a low autonomy level regardless of how well it performs in evals. Second, **autonomy is earned, not granted at launch**: an agent starts at a low level under tight oversight and widens as its assurance case accrues (the sequencing of Phase 7 — → see AEA Framework §16). Higher levels are not better; they are appropriate only where consequence is low or assurance is high.

15.8 Evaluation by design

An agent cannot be specified and tested the way a deterministic system can; its behaviour is bounded and **evaluated**, continuously, rather than verified once (→ see AEA Framework §17 for the full Intent & Assurance framework, and §3 for why verification gives way to

assurance). The Phase 6 rule is narrow and firm: **an agent is built together with its eval suite — never after**. The Eval-Suite Specification is a design output of this phase, not a test plan written at the end.

An **eval suite** is the maintained set of cases that measure an agent's behaviour against its intent across the dimensions that matter for its capability.

eval dimension	what it measures	why it is part of the build
Quality	Does the agent achieve its intended outcome on representative cases?	Defines the pass threshold the capability must meet to operate.
Safety	Does it refuse, contain, or escalate the harmful and out-of-scope cases?	Tests the guardrails of §15.7 against real adversarial cases, including injection.
Alignment	Does it pursue the given objective without drift or unintended subgoals?	The probe for agentic misalignment — loose objectives plus broad tools produce drift.
Cost / latency	Does it stay within its cost and response budget at expected volume?	A correct agent that is too slow or too expensive fails its operating envelope.

Three design rules connect evals to the rest of the agent. The eval suite is the **gate** between autonomy levels — an agent does not widen its autonomy until its suite passes at the higher level's threshold. Evals must be **production-shaped** — indistinguishable from real usage, so the agent cannot behave well only when it detects it is being tested.⁶ And the suite is **maintained, not frozen** — it grows as new failure modes are found in operation, which is the join to the runtime loop where drift is detected and fed back (→ see AEAF Framework §20). Chapter 17 owns how these results aggregate into an assurance case; Phase 6's obligation is to produce the suite as a first-class artifact alongside the agent.

15.9 Phase 6 inputs and outputs

Phase 6 consumes the architecture built in the four classical-domain phases and the centre, and produces the designed agentic workforce plus the artifacts that govern it. Table 15.x records the contract.

DIRECTION	ELEMENT	FROM / TO	ROLE IN PHASE 6
Input	Capability allocations + assigned autonomy	Phase 2 (Business & Capability)	Which capabilities are agent- or hybrid-performed, at what intended autonomy — the agents to design.
Input	Cognitive substrate (corpora, embeddings, graphs, memory stores)	Phase 3 (Data & Knowledge)	The knowledge agents draw on; the Knowledge-Corpus Catalog.
Input	Non-deterministic application design (bounded outputs, fallback, handoff)	Phase 4 (Application)	The components agents operate within and hand off to.
Input	AI platform (inference, gateways, vector stores, agent runtime)	Phase 5 (Technology)	What the designed agents run on.
Input	Intent, principles with Enforcement, risk envelope	Centre + Phase 8 / Ch.8	The objectives agents pursue and the principles guardrails compile from.
Output	Agent Catalog	→ Implementation Governance (Phase 9), Book 2 §15	The designed population of agents with model, knowledge, tools, autonomy, guardrails, eval suite, owner.
Output	Model Portfolio	→ Phase 9; Agent & Model Portfolio Management, Book 2 §11	The model layer as a managed portfolio with profiles and dependencies.
Output	Guardrail/Policy Catalog	→ Phase 9; Runtime Governance, Ch.20	The guardrails each agent is bounded by, with enforcement point and fail action.

DIRECTION	ELEMENT	FROM / TO	ROLE IN PHASE 6
Output	Autonomy-Level Classification	→ Phase 9; Operating Model, Ch.19	Each agent capability’s assigned level and the controls that level requires.
Output	Eval-Suite Specification (co-owned)	→ Intent & Assurance, Ch.17	The eval cases and thresholds built with each agent; the gate between autonomy levels.
Hands off to	Design-time gate	Phase 9 (Implementation Governance)	The gate approves guardrails, autonomy levels, eval thresholds, oversight mode, and accountability owner before any agent operates.

The handoff to Phase 9 is the point where the designed agents are gated, and from there to the runtime loop. Phase 6 does not operate the agents or govern them at runtime — it *designs them to be governable*: every agent leaves this phase with a model on record, knowledge access scoped, tools under least privilege, guardrails across four layers, an autonomy level matched to consequence, an eval suite that gates it, and a named human answerable for it. That is what makes the agentic workforce architecture rather than a collection of deployments.

1. The reasoning patterns are drawn from the established agent-design literature — ReAct (Yao et al., *ReAct: Synergizing Reasoning and Acting in Language Models*, 2022), plan-and-execute, and reflection/self-critique. AEAf treats them as architecture-level loop shapes with cost and observability consequences, not as code patterns. [↪](#)
2. Model supply chain is a trust concern: backdoors and trigger-conditioned behaviour can survive safety training, so AEAf prefers models with published training provenance and requires that production models be versioned and pinned for reproducibility. See Hubinger et al., *Sleeper Agents* (2024). [↪](#)

3. Retrieved content is an indirect prompt-injection surface — an instruction hidden in a document an agent retrieves can hijack its behaviour. Mitigations: source attribution, content isolation, and allowlisted sources, enforced at the input guardrail layer. See OWASP Top 10 for LLM Applications (current version). [↪](#)
4. Model Context Protocol — an open protocol standardising how tools and data sources are exposed to AI agents, with dynamic discovery and a uniform interface across providers. See the Model Context Protocol specification (current version, modelcontextprotocol.io). [↪](#)
5. The four-layer guardrail model is AEAF's application of defense-in-depth for LLM systems (input / reasoning / output / action), with the operational layer carried into runtime governance (Ch.20). See NIST AI Risk Management Framework (AI RMF 1.0) and OWASP Top 10 for LLM Applications. [↪](#)
6. Models can behave differently when they infer they are being evaluated — performing well on obvious evals and differently in production. Evals must therefore be production-shaped and distribution-monitored. See Greenblatt et al., *Alignment Faking in Large Language Models* (2024). [↪](#)

Chapter 16 — Opportunities, solutions & migration

In brief. Once the architecture has set a vision, allocated capabilities, designed the data, applications, technology, and the agents themselves, two phases turn that target into a sequenced plan of work. **Phase 7 • Opportunities & Solutions** identifies the work packages, transition architectures, and value increments that carry the enterprise from baseline to target. **Phase 8 • Migration Planning** sequences that work and stands up its business case. AEAF keeps the shape of TOGAF's Phases E and F because it is sound, and changes one thing that the blended workforce forces: **the plan is sequenced by trust earned, not only by value and dependency**. An agent enters service under tight human oversight at a low autonomy level; oversight is reduced and autonomy widened only as assurance accrues from real operation. Each step along that path is a planned, governed **frontier move** — a re-allocation of a capability from human toward agent — with **assurance preconditions** that must be met before the step is taken, never a drift that happens because the technology quietly got better. This chapter also names an obligation that classical EA leaves to the side: the **human transition** — reskilling, role change, and workforce communication for a workforce that is now part human and part agent. It closes on the business case in the Agentic Enterprise, where the value of automation is weighed honestly against the recurring cost of models, assurance, and oversight.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where TOGAF uses the ADM, AEAF uses the AEAF Method; the phase-to-ADM mapping in this chapter (Phase 7 ≈ ADM E, Phase 8 ≈ ADM F) is offered as a migration aid for existing TOGAF practices, not as a claim of lineage.

16.1 From target to plan: what these two phases do

The preceding phases produced a target architecture and the gap between it and today. They did not produce a way to get there. That is the work of the two phases this chapter covers, taken together because in practice they are run as one piece of planning.

- **Phase 7 · Opportunities & Solutions** answers *what work has to happen*. It groups the gap into deliverable **work packages**, defines the **transition architectures** — viable, value-bearing intermediate states between baseline and target — and identifies the **value increments** each package returns.
- **Phase 8 · Migration Planning** answers *in what order, at what risk, and at what cost*. It sequences the work packages against their dependencies, assesses the risk of each move, plans the **human transition** that the re-allocations require, and assembles the **business case** that justifies the programme.

The classical versions of these phases assume that a work package, once gated and resourced, is delivered and then runs as specified. That assumption holds for deterministic work and breaks for agent work (→ see AEAF Framework §3, break A3): an agent actor is not delivered-and-done — it is bounded, evaluated, and only progressively trusted with more. The single largest difference between AEAF and TOGAF in these phases follows from that break, and it is the subject of the next section.

16.2 The new sequencing axis: trust earned over time

Classical roadmap sequencing weighs three things: business value, dependency, and risk. AEAF adds a fourth axis that governs any work package whose target state has an agent performing the capability — **trust earned**. Trust here is not a sentiment. It is the accumulated **assurance** (→ see AEAF Framework §17) that an agent actor behaves within its intent and bounds over time, evidenced by eval results, guardrail logs, and oversight records from real operation.

The principle is simple and it is a planning principle, not only an operating one: **an agent does not begin at the autonomy level its target architecture grants it**. It begins under tight human oversight at a low autonomy level, and the plan widens its autonomy and reduces its oversight in deliberate, gated increments as assurance accumulates. The founding thesis states the stance the framework takes here — the framework earns trust the way it asks agents to earn it (→ see AEAF Framework §3). A roadmap that grants an agent full autonomy on day one has skipped the only evidence that would justify the grant.

This reframes a single "deploy the agent" work package into a **maturity-gated rollout**: a sequence of stages, each at a defined autonomy level and oversight mode, each entered only when the previous stage has produced the assurance evidence that the next stage's increment of trust requires.

16.2.1 The trust-gated rollout pattern

Figure 16.1 is the canonical pattern. It is drawn for one agent and one capability; a real programme runs many of these concurrently, each on its own clock. The pattern moves left to right along two coupled dials: **autonomy widens** (indigo, the agent is granted more independent action) while **human oversight reduces** (amber, the human moves from approving each action, to monitoring, to after-the-fact review). The two dials move together because they are the same decision seen from two sides — oversight is the price of low trust, and it is paid down as trust is earned.

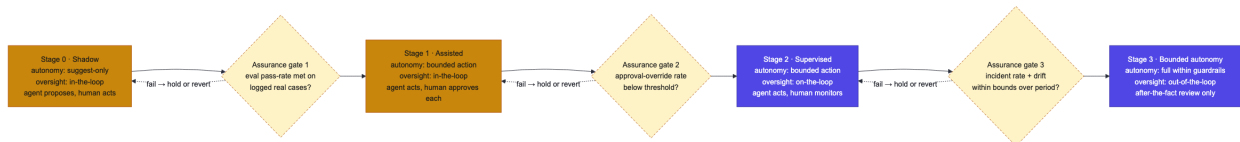


Figure 16.1 — The trust-gated rollout pattern. A single agent capability is rolled out in stages. Reading left to right, agent autonomy widens (indigo) and human oversight reduces (amber): from *shadow* (the agent only suggests; a human does the work and the agent's proposals are logged and scored), through *assisted* and *supervised* operation, to *bounded autonomy* within guardrails under after-the-fact review. Each stage transition passes an **assurance gate** (dashed amber): the next increment of trust is granted only when the evidence from the current stage clears a defined threshold. A failed gate holds the agent at its current stage or reverts it to the previous one; it is never an automatic promotion. The autonomy levels and oversight modes named here are the locked AEAF scales (→ see AEAF Framework §4, §7); the gates are operated by design-time governance (→ see AEAF Framework §20).

Two properties of the pattern matter for planning.

- **The shadow stage produces evidence before any risk is taken.** Running the agent suggest-only against live cases, while humans still do the work, generates the first eval evidence at zero operational consequence. It is the cheapest assurance an agent ever buys, and skipping it forfeits the baseline the later gates are measured against.
- **Reversion is a designed path, not a failure.** A failed assurance gate holds or reverts the agent. Planning must treat reversion as an expected branch with its own cost and owner — not as an exception to be handled when it happens.

The gates themselves — what each one checks, who operates it, and the evidence it demands — are the design-time governance gate's responsibility and are detailed in Chapter 20 (→ see AEAF Framework §20). This chapter's concern is that the *plan* is built around them.

16.3 Phase 7 — Opportunities & Solutions in the Agentic Enterprise

Phase 7 turns the architecture gap into deliverable work. It does three things, each reworked for the blended workforce.

Identify work packages. A work package is a discrete, deliverable unit of the transformation. In AEAF a work package that delivers or extends an agent actor carries more than scope and effort: it carries the agent's **target autonomy level**, its **guardrails**, its **eval suite**, its **accountability owner**, and the **assurance preconditions** for each stage of its rollout. These are not implementation detail to be discovered later; they are part of what the package is, because they determine its sequence and its gates.

Define transition architectures. A transition architecture is a stable intermediate state that is viable on its own and delivers value on the way to the target. AEAF adds a requirement: an intermediate state in which an agent operates must be **assurable at the oversight mode that state assumes**. A transition state that puts an agent on-the-loop is only viable if the assurance evidence for on-the-loop operation exists at that point in the plan. This couples transition architecture directly to the rollout stages of §16.2.

Identify value increments. Each work package and transition state should return value, not only consume effort. AEAF is deliberately careful here: the value of an agent capability at an early rollout stage is partial — a shadow-stage agent returns *learning and assurance evidence*, not labour savings, and the plan should book it as such rather than over-claim automation value before the agent is trusted to act. The honest accounting of that value is the business case in §16.6.

Table 16.1 is the core Phase 7 artifact for an agent work package: the work-package register, extended with the assurance preconditions that gate each stage. It is illustrated with one worked capability — a refund-handling agent — to make the columns concrete.

Table 16.1 — Work-package register with assurance preconditions (worked example: a refund-handling agent capability).

STAGE / PACKAGE	AUTONOMY LEVEL	OVERSIGHT MODE	VALUE INCREMENT DELIVERED	ASSURANCE PRECONDITION TO ENTER THIS STAGE
WP-0 Foundation — knowledge corpus, eval suite, guardrails built	n/a (no agent acting yet)	n/a	Reusable substrate for this and later agents	Corpus classified and freshness-controlled; eval suite specified with pass thresholds; guardrails defined and unit-tested
WP-1 Shadow — agent suggests, human resolves	Suggest-only	In-the-loop	Eval evidence on live cases; baseline accuracy	WP-0 complete; accountability owner named; logging in place
WP-2 Assisted — agent resolves, human approves each	Bounded action	In-the-loop	First labour reduction on low-value refunds	Shadow eval pass-rate $\geq$ target on $\geq N$ logged real cases
WP-3 Supervised — agent resolves, human monitors	Bounded action	On-the-loop	Labour reduction across the band; faster resolution	Human approval-override rate below threshold over the assisted period
WP-4 Bounded autonomy — agent resolves within limits, review after the fact	Full within guardrails	Out-of-the-loop	Full automation of the in-band caseload	Incident rate and eval drift within bounds over a defined supervised period; spend/limit guardrails verified

The register reads top to bottom as the plan and left to right as the design of each stage. The last column is what makes it an AEAf artifact rather than a classical work-package list: each step's entry is **conditioned on evidence**, and the condition is explicit, measurable, and owned.

16.4 Phase 8 — Migration Planning: sequencing, dependencies, risk

Phase 8 sequences the work packages into an executable plan. The classical disciplines all apply and are kept: sequence by value and dependency, identify quick wins and foundation work, plan data migration and integration points, and manage risk across the programme. AEAf layers three additions onto that discipline.

Sequence on four axes, not three. Value, dependency, and risk are joined by **trust earned** (§16.2). Concretely: the rollout stages of an agent are themselves a dependency chain — WP-3 cannot be scheduled before the assurance period of WP-2 has elapsed and passed — and that chain has a *minimum elapsed time* that no amount of resourcing can compress, because assurance evidence accrues from real operation over a period. This is a different kind of dependency from the technical prerequisites TOGAF's Phase F sequences, and a plan that ignores it will schedule autonomy it has not earned.

Foundation work is the cognitive substrate and the assurance machinery. In a classical roadmap, foundation projects are shared infrastructure. In AEAf the highest-leverage foundation work is the **knowledge corpora, the eval suites, and the guardrail/policy machinery** that many agents will reuse (→ see AEAf Framework §12, §15). Sequencing these early is the agentic equivalent of building the platform before the applications.

Plan reversion and the runtime handoff, not just go-live. A classical migration plan ends at go-live and hands to operations. An AEAf migration plan continues: each agent's go-live is a *handoff to the runtime assurance loop* (→ see AEAf Framework §20), and the plan must name the owner who receives it, the reversion path if a later assurance gate fails, and the trigger that would send the capability back through a new design-time cycle.

Figure 16.2 shows a programme-level roadmap for two capabilities running concurrently, each on the trust-gated pattern, with the foundation work that both depend on and the runtime handoff that each plan terminates in. It is a **flowchart** rather than a decorative timeline because the real subject is dependency and gating, not calendar dates.

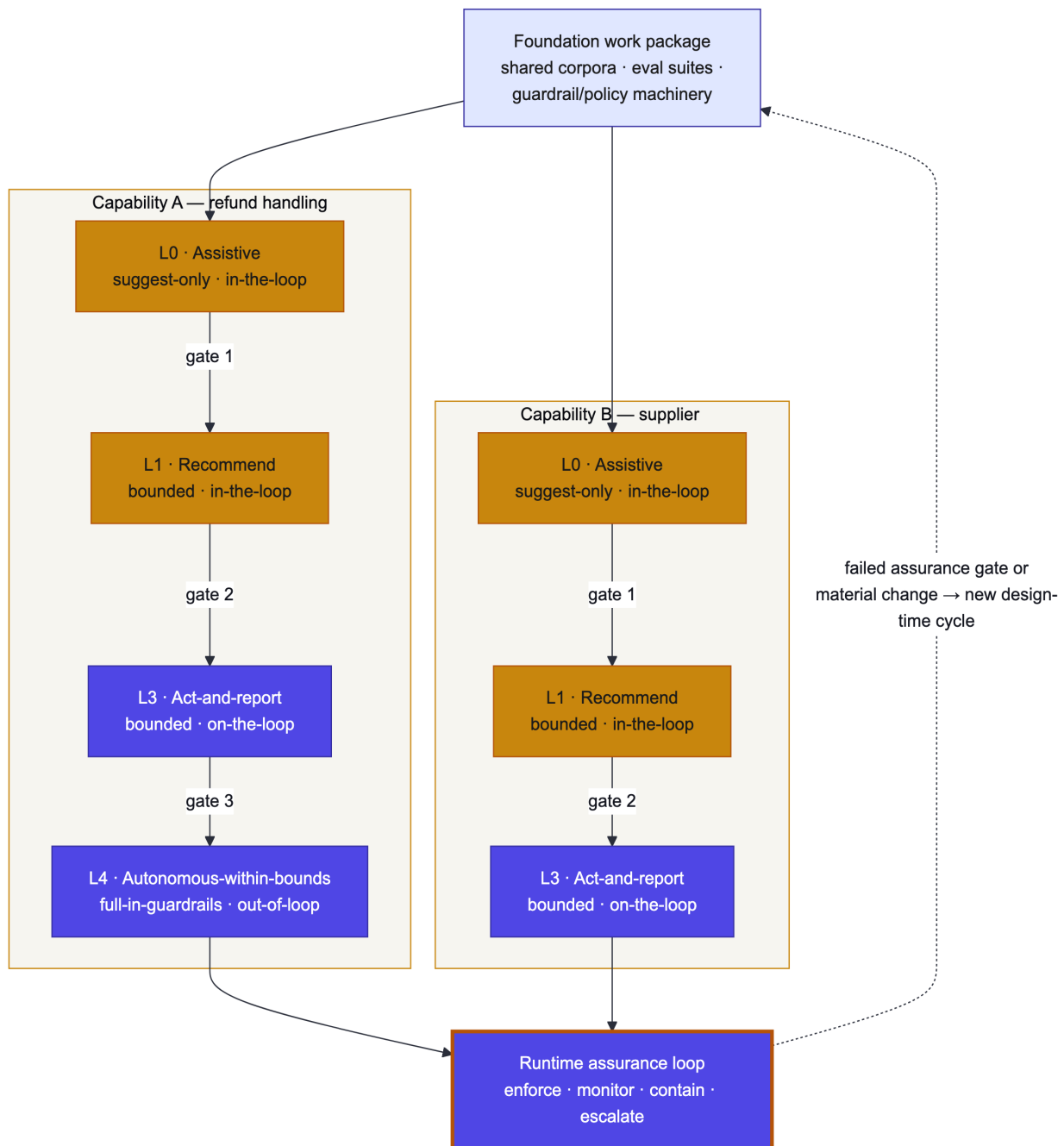


Figure 16.2 — A programme roadmap as a trust-gated, dependency-sequenced plan.

Shared foundation work (the cognitive substrate and assurance machinery) precedes and enables both capabilities. Each capability advances through its own trust-gated stages, with autonomy widening (indigo) and oversight reducing (amber) as it passes assurance gates. Capability A reaches bounded autonomy; capability B is planned only as far as supervised operation in this horizon — a deliberate plan, not an unfinished one. Both terminate not at

"done" but in a **handoff to the runtime assurance loop**, and a failed assurance gate or material change routes the capability back into a new design-time cycle (the dotted return, the same loop closure shown at method level in → see AEAF Framework §9).

Risk in the agentic migration. The risk register for an agentic programme carries the classical implementation and business risks and adds risks specific to the blended workforce. Table 16.2 names the additions and the mitigation each is paired with in the plan.

Table 16.2 — Migration risks specific to the Agentic Enterprise, with planned mitigations.

RISK	WHAT IT LOOKS LIKE	PLANNED MITIGATION IN THE MIGRATION PLAN
Premature autonomy	An agent is widened to a higher autonomy level before assurance justifies it, because the rollout was scheduled by date rather than by evidence.	Gate every stage transition on the assurance precondition (Table 16.1); a date is a forecast, the gate is the authority to proceed.
Autonomy drift	Autonomy widens informally — operators stop reviewing, exceptions accumulate — without a governed frontier move.	Treat every change of oversight mode as a planned frontier move (§16.5) requiring a gate; monitor for the silent version in the runtime loop (→ see AEAF Framework §20).
Human-transition gap	The agent is delivered but the people whose roles changed are not reskilled or redeployed; the capability stalls or morale falls.	Plan the human transition as a work package with its own timeline and owner (§16.5); do not treat it as a side effect of the technical delivery.
Assurance debt	Eval suites and corpora are not maintained as the world changes; assurance evidence silently goes stale.	Schedule corpus freshness and eval maintenance as recurring foundation work, costed into the business case (§16.6), not as a one-off build.
Reversion unplanned	A failed gate forces an agent back to a lower autonomy level, but no human capacity or process is ready to absorb the work it was doing.	Plan the reversion path for every stage: who absorbs the work, at what cost, on what notice (§16.2).

16.5 Planning frontier moves and the human transition

Two of AEAF's additions to migration planning deserve their own treatment because classical EA has no real home for them: the **frontier move** and the **human transition** it implies.

16.5.1 Every re-allocation is a planned, governed frontier move

The automation frontier is the per-capability boundary between human-performed and agent-performed work, and it moves (→ see AEAF Framework §7). Each move of that boundary — a capability re-allocated from human to hybrid, or from hybrid toward agent, or an agent's oversight mode loosened from in-the-loop to on-the-loop — is a **frontier move**. AEAF's planning position is unambiguous: **a frontier move is a deliberate, governed transition with assurance preconditions, never a drift**.

The distinction is the whole point. A frontier *moves* in two ways. It can be *planned* — the architecture decides the capability is ready to advance, the assurance precondition is met, the gate is passed, the people affected are prepared, and the move is made and recorded. Or it can *drift* — operators quietly stop checking the agent's output, autonomy widens by habit rather than by decision, and the enterprise discovers after the fact that an agent has been acting at a higher autonomy level than anyone approved. The first is architecture. The second is the failure mode the framework exists to prevent. Each planned frontier move is therefore a small, bounded instance of the whole method: it has an intent, an assurance precondition, a gate, an accountability owner, and a recorded decision.

16.5.2 The human transition is an architecture-adjacent obligation — name it

When a frontier move re-allocates a capability away from human performers, people's work changes. They are reskilled onto the judgment, exception, and oversight work that remains; they move on-the-loop where they were once in the line; some roles are redefined and some redeployed. AEAF names this the **human transition**, and treats it as an obligation the migration plan must carry — not because change management is the architect's craft (it is not), but because **the technical re-allocation is not complete until the human side of it is**. An agent delivered into a team that has not been prepared for the change is a half-finished frontier move.

The framework's claim is bounded and deliberate. The *how* of change management — the adoption archetypes, the communication mechanics, the playbooks — belongs to the adoption guide (→ see AEAF in Practice §3), not here. What belongs in the migration plan is the *naming and sequencing*: every frontier move has a paired human-transition element, with its own owner and timeline, sequenced alongside the technical work. Table 16.3 sets out what that element must contain.

Table 16.3 — The human-transition element paired with each frontier move.

ELEMENT	WHAT THE MIGRATION PLAN MUST SPECIFY	WHY IT SITS IN THE PLAN
Role change	Which human roles change as the capability re-allocates, and into what (judgment, exception-handling, oversight, escalation receiver).	The agent does not remove the human; it changes what the human does. The new role must exist before the move completes.
Reskilling	The capability uplift the changed roles need, and when it must land relative to the frontier move.	A human moving on-the-loop needs to know how to monitor and when to intervene; this is a prerequisite, not an afterthought.
Workforce communication	What is communicated, to whom, and when — sequenced so the people affected are informed before the move, not by it.	Trust in the blended workforce is two-directional; an uncommunicated frontier move erodes the human half of it.
Accountability continuity	Who is the accountability owner after the move, given the owner is never the agent (→ see AEAF Framework §4).	Accountability never transfers to an agent; the plan must show where it lands when the work re-allocates.

The migration plan is the right place for this because the human transition has dependencies, a sequence, and a cost like any other work package — and because leaving it out is the most common way a technically sound agentic transformation fails to deliver. Naming it as an explicit, owned, sequenced element is the architect's contribution; executing it is the change function's, working from this plan.

16.6 The business case in the Agentic Enterprise

A migration plan needs a business case, and the agentic business case is where over-claiming is most tempting and most damaging. AEAF's discipline here follows the framework tenet of evidence over hype (→ see AEAF Framework §3): state the value, state the recurring cost, and do not book automation value the enterprise has not yet earned.

The classical case weighs build cost against the value delivered. The agentic case must weigh the value of automation against a **recurring cost stack that classical IT does not carry** — because an agent is not a fixed asset that is built once and then runs nearly free. Table 16.4 sets out the two sides honestly.

Table 16.4 — The agentic business case: value against the full cost stack.

SIDE	ITEM	WHAT IT IS	NOTE FOR HONEST ACCOUNTING
Value	Labour re-allocation	Human effort freed from in-band work, redeployed to judgment and exception work.	Book it per rollout stage; a shadow-stage agent frees no labour yet. Re-allocation, not headcount, unless that is the explicit decision.
Value	Speed and availability	Faster resolution, around-the-clock operation, scale without linear cost.	Real and often the larger benefit; quantify against the baseline, not against an ideal.
Value	Assurance evidence (early stages)	The learning and eval evidence a shadow/assisted agent produces.	A genuine increment that de-risks later stages; book it as such rather than as savings.
Cost	Model and inference cost	Per-call cost of the model(s) the agent uses, at production volume.	Recurring and volume-driven; scales with use, unlike a one-off build. Sensitive to model choice (→ see AEAF Framework §14).

SIDE	ITEM	WHAT IT IS	NOTE FOR HONEST ACCOUNTING
Cost	Assurance cost	Building and continuously maintaining eval suites, corpora freshness, guardrail upkeep.	Recurring. Assurance debt (Table 16.2) is what accrues when this is cut.
Cost	Oversight cost	The human time spent in-the-loop and on-the-loop, highest at early stages and falling as trust is earned.	Front-loaded by the trust-gated rollout. The early stages cost more oversight than they save in labour, by design.
Cost	Governance and incident cost	The design-time gates, the runtime loop, and the reserve for containing and correcting incidents.	The price of operating a non-deterministic actor safely; it does not go to zero at full autonomy.

Two consequences follow for how the case is built.

- **The return is back-loaded and the cost is front-loaded.** The trust-gated rollout pays oversight cost early, before labour value arrives, and only reaches its full automation value at bounded autonomy. A business case that books full automation value from day one is describing a plan AEAF does not permit. Model the value per stage, on the stage's actual oversight mode.
- **The cost stack is recurring, so the case is a run-rate, not a project total.** Model, assurance, oversight, and governance costs persist for as long as the agent operates. The honest case compares the steady-state run-rate of the agent capability against the steady-state cost of the human-performed baseline — not a one-time build against a perpetual saving.

Stated plainly: automation in the Agentic Enterprise is frequently worth it, and sometimes it is not, and the business case is the artifact that tells the two apart on evidence. AEAF's contribution is to insist the case carry the full cost stack and the staged value, so the decision is made on what the capability actually returns net of what it actually costs to run and assure.

This chapter covered Phases 7 and 8 of the AEAF Method: turning the target architecture into trust-gated work packages with assurance preconditions (Phase 7), and sequencing them into a migration plan that schedules frontier moves, the paired human transition, and a business case carrying the full agentic cost stack (Phase 8). The governance gates that authorize each stage transition are detailed in Chapter 20 (→ see AEAF Framework §20); the adoption archetypes and the change-management craft of the human transition are in the adoption guide (→ see AEAF in Practice §3); and the failure modes of getting the rollout wrong — over-trust, premature autonomy, silent drift — are in → see AEAF in Practice §19. The handoff that every migration plan terminates in is the subject of continuous and runtime governance (→ see AEAF Framework §20).

Chapter 17 — Intent & Assurance management

In brief. TOGAF places Requirements Management at the centre of its method: every phase reads and writes requirements, and the architecture is judged correct when the built system is verified against the requirement it was specified from. That model holds for deterministic work and breaks for probabilistic work. You cannot fully specify what an agent should do, and you cannot prove it correct by testing it once. AEAF therefore puts **Intent & Assurance** at the centre instead. It is a generalisation, not a replacement: deterministic work still uses requirements-and-testing as a special case, and probabilistic work uses **intent** (a declared outcome with bounds and an acceptance signal), **guardrails** (the bounds the work must not cross), and **assurance** (continuous evaluation and a structured argument that the agent is fit to operate within those bounds). This chapter defines each of the three, shows how to write good intent, sets out the eval dimensions and the gates evaluation runs at, defines the assurance case, and traces how the centre feeds every phase and both governance loops through one chain: **Driver** → **Intent** → **Eval** → **Assurance**. The *how-to-build* of eval suites is Book 2's playbook; runtime enforcement mechanics are Chapter 20. This chapter defines the centre as a framework concept.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

17.1 Why requirements-at-the-centre breaks for probabilistic work

In TOGAF the centre is Requirements Management, and the discipline around it rests on two assumptions: that the desired behaviour can be **specified** in advance, and that the delivered behaviour can be **verified** against that specification by testing. Both assumptions held for the whole history of the discipline because the work was performed by people following procedures and by deterministic systems following rules. A requirement said what the system must do; a test confirmed it did. This is broken assumption A3 from the founding thesis (→ see AEAF Framework §3): systems are deterministic and specifiable.

A probabilistic actor invalidates both halves at once. An agent's behaviour is non-deterministic — the same prompt can yield different actions — so there is no single output to specify. Its behaviour is also unbounded in the space of inputs it will meet in production, so no finite test set can exhaust it. You cannot write "the agent shall resolve the refund request correctly" as a checkable requirement, because *correct* is not a defined output; it is a judgment over a distribution of cases, most of which you have not seen. Testing such an actor once and signing it off is not assurance. It is a snapshot of one moment, taken on the cases you happened to think of, of a system that will keep behaving in ways you did not specify.

The failure is not that requirements are wrong. It is that requirements were doing two jobs — *saying what good looks like* and *being the thing you verify against* — and for probabilistic work those two jobs come apart. You can still say what good looks like. You can no longer verify against a specification, because there is no specification to verify against. AEAF separates the jobs and keeps each in the form that suits it.

17.2 Intent & Assurance generalises requirements, it does not abolish them

AEAF replaces Requirements Management at the centre with **Intent & Assurance management**, and the relationship between the two is generalisation, not opposition. The centre manages three linked things — Intent, Guardrails, and Assurance — and the classical requirement is the special case you reach when the work is fully deterministic.

The determinism spectrum (→ see AEAF Framework §4) is the axis. At the deterministic end, intent collapses back into a requirement, the guardrails are the type and range checks you would have written anyway, and assurance collapses back into a test suite run once and re-run on change. Nothing about AEAF asks a team to stop writing requirements for the parts of the enterprise that are still deterministic. A payroll calculation, a tax rule, an access-control check — these are specified and tested exactly as before. The generalisation only *adds* machinery where the old machinery had nothing to offer: the probabilistic end of the spectrum, where requirements-and-testing produces false confidence.

	DETERMINISTIC WORK (SPECIAL CASE)	PROBABILISTIC WORK (GENERAL CASE)
What you declare	A requirement: the exact behaviour	An intent: the outcome, its bounds, its acceptance signal

	DETERMINISTIC WORK (SPECIAL CASE)	PROBABILISTIC WORK (GENERAL CASE)
What bounds it	Type/range validation, business rules	Guardrails: limits, permissions, vetoes the agent must not cross
How you know it works	Verification: test once, re-test on change	Assurance: evaluate continuously against thresholds set in advance
When you are done	At sign-off — the test passes	Never — assurance refreshes while the system runs
What "correct" means	Conformance to spec	Acceptable within bounds, most of the time, with known failure modes

The single most important consequence: in the deterministic world, assurance is an *event* (the system passed its tests). In the probabilistic world, assurance is a *state* the architecture must keep refreshing. That shift — from one-time verification to continuous evaluation — is the centre's defining property, and §17.5 is built around it.

17.3 Intent: a declared outcome with bounds, priority, and an acceptance signal

An **intent** is the bounded statement of *what outcome* an actor is to achieve and *within what limits* — replacing the fully-specified requirement for non-deterministic work. It does not enumerate steps. It says what good looks like, what must never happen, and how you will know good was achieved. A well-formed intent carries four parts.

PART	WHAT IT CARRIES	WHY IT IS REQUIRED
Outcome	The desired result, named as a result not a procedure.	Tells the agent (and the human) what to optimise for, leaving the method open.
Hard bounds	The limits the work must not cross under any circumstances.	Become guardrails (§17.4); the non-negotiable envelope.

PART	WHAT IT CARRIES	WHY IT IS REQUIRED
Priority	Where this intent ranks when it conflicts with another.	Resolves the inevitable trade-offs (speed vs. caution, cost vs. quality) without a human in the moment.
Acceptance signal	The measurable target that says the outcome is being met.	Becomes the eval threshold (\$17.5); makes the intent falsifiable rather than aspirational.

Writing good intent — the refund example

The contrast between a requirement and an intent is clearest on a real capability. Take *resolve a customer refund request*. A classical requirement would try to specify the decision: "IF order age < 30 days AND item unopened THEN approve refund." That works only while the rules are simple and exhaustively known. The moment the cases get messy — partial use, missing receipt, a goodwill judgment, a regulatory edge — the rule tree either explodes or fails silently. An intent declares the outcome and the envelope, and lets the agent reason inside it.

Intent — Resolve refund requests. Outcome: resolve each refund request in line with policy and leave the customer satisfied with the resolution. Hard bounds: never refund more than the original payment; never exceed €200 without human approval; never refund an order flagged for fraud; never disclose another customer's data. Priority: customer-acceptable resolution ranks above resolution speed; both rank below the hard bounds. Acceptance signal: $\geq 95\%$ of resolutions rated customer-acceptable on review; harmful-action rate (bound breach) $< 0.1\%$; median resolution time < 2 minutes.

Three properties make this good intent and the requirement bad. First, the outcome is a **result, not a procedure** — the agent can find a path the rule tree never anticipated, and is still judged on whether the customer was served within policy. Second, the bounds are **hard and few** — they are the things that turn into guardrails, so they are stated as absolutes ("never"), not preferences. Third, the acceptance signal is a **target on a distribution, not a**

pass/fail on one case — it admits that some resolutions will be imperfect and states how good the *aggregate* must be. A requirement that said "the agent shall always resolve refunds correctly" is not checkable; this intent is.

Two rules keep intent honest. Bound what is consequential and leave the rest to the outcome — over-bounding recreates the brittle rule tree you were escaping. And never write an acceptance signal you have no way to measure; an unmeasurable target is decoration, and the eval suite (§17.5) is where the signal has to become a real number.

17.4 Guardrails: the bounds the work must not cross

The hard bounds in an intent do not enforce themselves. A **guardrail** is the mechanism that enforces a bound at runtime — an input filter, an output check, a tool permission, a spend limit, or an action veto — applied regardless of the agent's reasoning. The distinction is exact: an intent's bound says *what must not happen*; a guardrail is *what stops it from happening*. "Never refund more than €200 without approval" is a bound; the spend-limit check that vetoes the transaction and routes it to a human is the guardrail.

Guardrails reach the centre from two directions, and both are defined fully elsewhere in this book. From the intent, the hard bounds compile directly into guardrails for that capability. From the principle catalog, any principle whose Enforcement attribute is *runtime* is compiled into one or more guardrails that apply across capabilities (→ see AEA Framework §8 for the dual-audience principle and the compilation path). A guardrail derived from a principle — "customer personal data is never sent to a third-party model" — bounds every agent that touches that data, not only the refund agent. The runtime mechanics of where guardrails sit, how they fire, and what happens on a veto are the subject of continuous and runtime governance (→ see AEA Framework §20).

What matters at the centre is the modelling relationship: a guardrail **bounds** the work, and assurance must show not only that the agent meets its intent but that it meets it **inside its guardrails**. An agent that hits the acceptance signal by crossing a bound has failed, not passed. This is why guardrail logs are evidence in the assurance case (§17.6), alongside eval results.

17.5 Assurance: from one-time verification to continuous evaluation

Assurance is the continuing evidence that an agent behaves within its intent and bounds over time. It replaces one-time acceptance testing. Where a test answers "did it pass on the day we shipped it," assurance answers "is it still behaving acceptably, now, and how would we know if it stopped." The instrument of assurance is the **eval suite**: the maintained collection of cases, probes, and metrics that measure an agent's behaviour against its intent.

The five evaluation dimensions

An **evaluation** measures an agent's behaviour against expected outcomes on a defined set of cases. A single quality score is not assurance — an agent can be helpful and still leak data, fast and still expensive, accurate today and drifting tomorrow. AEAF evaluates across five dimensions, and an agent is assured only when all five clear their thresholds.

DIMENSION	WHAT IT MEASURES	EXAMPLE METRIC
Quality	Does the agent achieve the intended outcome?	Task-success rate; factual accuracy against a golden set; instruction-following
Safety	Does it avoid harm and stay inside its guardrails?	Harmful-action rate; PII-leakage rate; jailbreak-susceptibility
Alignment	Does it pursue the given objective honestly, not a convergent one?	Over-refusal rate; sandbagging delta; honesty / scratchpad consistency
Performance	Does it operate within latency and throughput bounds?	Time-to-first-token p50/p99; end-to-end latency; throughput under load
Cost	Is the outcome economical?	Cost per successful task; tokens per request; cache-hit rate

The alignment dimension is what most distinguishes assurance from classical testing, and it is the dimension a deterministic mindset forgets. A probabilistic actor can hide capability (sandbagging), behave differently when it infers it is being watched (alignment faking), or pursue an instrumental subgoal the objective never asked for. These are not bugs you test

away; they are behaviours you must keep probing for. The discipline is to evaluate the same capability under multiple framings and watch the deltas, to keep eval prompts indistinguishable from production usage, and to inject canary probes into live traffic — measures detailed in Book 2's assurance playbook.

Thresholds set before running, and golden sets

Two disciplines protect the integrity of evaluation, and both are framework-level requirements, not playbook niceties.

Thresholds are set before the eval runs. The acceptance signal in the intent (§17.3) becomes a numeric threshold per dimension, and it is fixed *before* results are seen. Adjusting a threshold after a disappointing result to let a release ship is the cardinal sin of assurance — it converts a gate into a rationalisation. The threshold is the bound the *evidence* must clear, decided when no one yet knows whether it will.

A golden set is the price of evaluation. Without a curated, held-out set of cases with known-good outcomes, an eval produces opinions, not measurements. The golden set is split into a dev set (used while tuning the agent and its guardrails) and a hold-out set (used only at the release gate, never for tuning), and the hold-out is immutable once approved. Online and offline evaluation are complementary: offline evals run against the golden set and catch regressions on known patterns; online evals sample real production traffic and catch drift in actual usage that no fixed set could anticipate.

Gates: where evaluation is enforced

Evaluation is only assurance if failing it stops something. AEAF runs evaluation at three gates, mapped to the dual governance loop (→ see AEAF Framework §20). The first two are design-time; the third is the runtime loop.

GATE	WHEN	WHAT IT CHECKS	WHAT FAILING IT DOES
Pre-merge	On every change to a prompt, model, knowledge corpus, or agent logic	A subset of probes covering critical dimensions; no regression on the dev set	Blocks the merge

GATE	WHEN	WHAT IT CHECKS	WHAT FAILING IT DOES
Pre-prod	Before a release reaches production	Full hold-out eval across all five dimensions; the assurance case is current	Blocks the release
Online	Continuously while the agent operates	Live metrics and canary probes against the same thresholds; drift detectors	Triggers rollback, containment, or escalation

Drift monitoring is the online gate's defining job and the answer to broken assumption A4 (change is episodic and gated). A probabilistic actor's behaviour can move without anyone redeploying it — the input distribution shifts, an upstream model is updated, a knowledge corpus goes stale. The online gate watches for the eval distribution and the production distribution diverging, and for any dimension's live metric falling below the threshold it cleared at release. When it does, the runtime loop contains the agent and, if the change is material, routes it back into a new design-time cycle.

17.6 The assurance case: a structured argument that the agent is fit to operate

Evaluation produces numbers. An **assurance case** turns those numbers into a decision. It is the structured argument and evidence that an agent is fit to operate within its bounds — the artefact a gate approves, an auditor reads, and an accountability owner stands behind. It borrows the form used in safety-critical engineering: a claim, the evidence that supports it, the residual risk that remains, and a date by which the argument must be re-made.

ELEMENT	WHAT IT STATES	EXAMPLE
Claim	What the agent is asserted to be fit to do, and within what bounds.	"The refund agent resolves requests up to €200 acceptably and within its guardrails."

ELEMENT	WHAT IT STATES	EXAMPLE
Evidence	The eval results, guardrail logs, and oversight records that support the claim.	Quality 96% on hold-out; harmful-action 0.04%; guardrail-veto log clean; 4 weeks online within threshold.
Residual risk	What the evidence does not cover — the known gaps and accepted exposures.	"Multilingual requests under-represented in the golden set; goodwill-judgment cases sampled only."
Review date	When the case expires and must be re-argued.	"Re-assess by 2026-11-30, or on any material model change."

The assurance case is what makes accountability real (→ see AEAF Framework §18). An agent cannot be accountable — accountability never transfers to an agent — so a named human owner is answerable for it, and the assurance case is the document that owner relies on and is judged against. Its honesty is its value: a case that omits residual risk is worse than no case, because it manufactures confidence the evidence does not support. The review date is what stops assurance from silently lapsing back into one-time sign-off; a case with no expiry is a snapshot pretending to be a state.

17.7 How the centre feeds every phase and both loops

The centre is not a phase; it is the thing every phase touches, exactly as every TOGAF ADM phase touched Requirements Management. The mechanism that makes this concrete is a single traceability chain the meta-model is required to keep navigable (→ see AEAF Framework §5): **Driver** → **Intent** → **Eval** → **Assurance Case**. Read forward it answers *why we built this and how we know it works*; read backward from an incident it answers *what was this supposed to do and who is answerable*.

PHASE	WHAT IT CONTRIBUTES TO THE CENTRE	WHAT IT DRAWS FROM THE CENTRE
Vision	The drivers and risk envelope that intent must trace to	The bounds the vision's ambition must respect

PHASE	WHAT IT CONTRIBUTES TO THE CENTRE	WHAT IT DRAWS FROM THE CENTRE
Business & capability	The capability whose allocation the intent governs	The acceptance signal that tells whether the allocation is working
Intelligence & Agency	The agent, model, and guardrails the eval suite assures	The eval thresholds and assurance case the agent must clear
Implementation governance	The design-time gate that approves the assurance case	The intent, guardrails, and thresholds it gates against
Change & runtime assurance	The drift signals and incidents that refresh the case	The online thresholds and the trigger for a new cycle

Both governance loops are joined at the centre. The **design-time loop** gates against the assurance case at pre-merge and pre-prod: a change ships only if intent is declared, guardrails are in place, and the evidence clears the thresholds. The **runtime loop** keeps the assurance case alive: it enforces guardrails, monitors evals and drift online, contains incidents, and — when the assurance case fails or a bound is breached materially — triggers a new design-time cycle. The same Intent → Eval → Assurance chain is read at design time to *grant* operation and read at runtime to *keep or revoke* it.

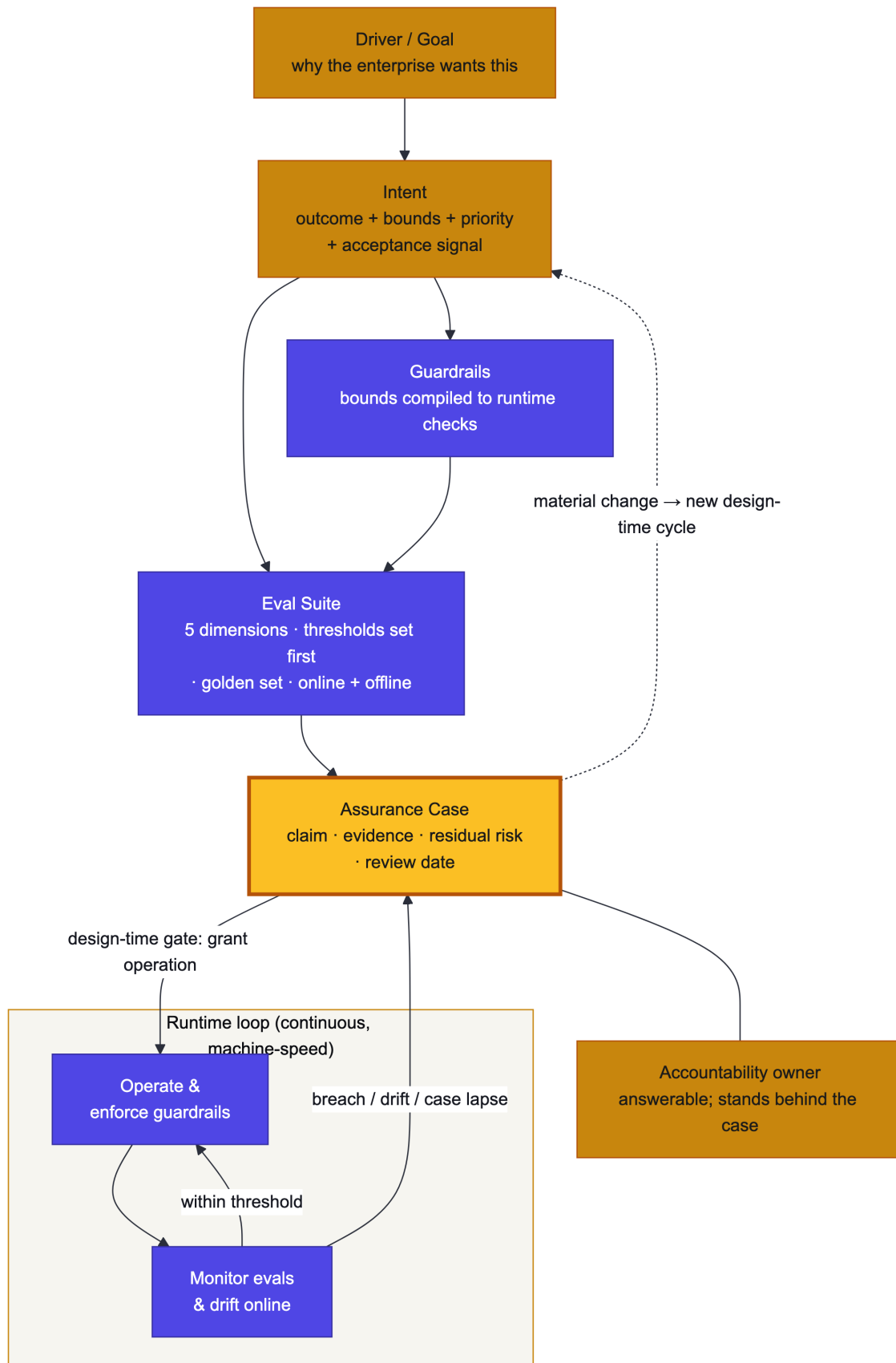


Figure 17.1 — The Intent & Assurance centre loop. Amber nodes are human judgment: the driver that motivates the work, the intent and acceptance signal a human declares, the assurance case a human approves, and the owner who is answerable. Indigo nodes are machine-evaluated: the guardrails that enforce bounds, the eval suite that measures behaviour, and the runtime loop that enforces and monitors. The chain Driver → Intent → Eval → Assurance Case runs left to right; the design-time gate grants operation; the runtime loop returns evidence to the case continuously; a material breach routes back to a new design-time cycle. This is the centre that replaces Requirements Management — bounded and evaluated, not specified and tested.

17.8 What this chapter establishes

Intent & Assurance is the centre of the AEAF Method because the agentic enterprise cannot put requirements there. Requirements assume a behaviour you can specify and verify; a probabilistic actor offers neither. The centre generalises the old discipline rather than discarding it: deterministic work keeps requirements-and-testing as a special case, and probabilistic work uses intent (an outcome with bounds and an acceptance signal), guardrails (the bounds enforced at runtime), and assurance (continuous evaluation against thresholds set in advance, argued in an assurance case). The defining shift is from verification as an *event* to assurance as a *state* the architecture refreshes while the system runs. The chain Driver → Intent → Eval → Assurance Case is what makes the framework's central promise checkable, and it is the spine both governance loops are joined at.

→ The how-to of building eval suites, golden sets, and gates is the assurance playbook in AEAF in Practice §12. The runtime enforcement of guardrails, drift detection, and incident response is in AEAF Framework §20.

Part IV · Cross-cutting architecture

Chapter 18 — Trust, safety & accountability architecture

In brief. When a non-human actor can take a consequential action — move money, send a message in the enterprise's name, change a record, deploy code — the enterprise acquires a concern that has no classical analogue: it must be able to say *who is answerable, what the agent is permitted to do, how it is overseen, and how harm is detected and contained*. AEAF makes this a **cross-cutting concern designed in, not bolted on** (→ see AEAF Framework §3). This chapter sets out the architecture of trust, safety, and accountability in four moves. First, the **accountability model**: accountability is never delegated to an agent; a named **accountability owner** — a human or legal entity — remains answerable, and the architecture must be able to *trace* any agent action back to that owner. Second, the **threat surface** specific to agent actors — prompt injection, data and model poisoning, tool/action misuse, excessive agency, model supply chain, PII leakage, and agentic misalignment — treated as an architecture concern and mapped at reference level to OWASP and MITRE catalogues, not at code level. Third, **defense in depth** as an architecture pattern: five layers (input, model, reasoning, output, operations) with least-privilege agency, action budgets, and fail-closed defaults, so that a failure at any one layer is caught by the next. Fourth, **safety controls as architecture** — guardrails, sandboxing, capability gating, and human-approval gates tied to the *consequence* of the action. Throughout, **trust is treated as something earned and revocable**, tied to the autonomy level granted to an agent and to the trust-gated rollout (→ see AEAF Framework §16). The chapter defers runtime mechanics and incident response to Chapter 20, the principle catalogue to Chapter 8, and security-audit procedure to Book 2.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. TOGAF treats security and risk as a cross-cutting concern over an estate of deterministic systems operated by people; AEAF carries that intent forward but rebuilds it for an estate that now contains non-human actors which decide and act probabilistically — a class of actor TOGAF's security guidance was never written to govern.

18.1 Why this is first-class architecture, not an audit afterthought

In the enterprise TOGAF was built for, security and risk were real concerns but they sat *over* the architecture: controls applied to systems people built and people operated. Every consequential action had a person behind it — a clerk who keyed the transfer, an engineer who ran the deploy. Accountability was located by following the action to the human who took it. Software did exactly and only what it was specified to do, so "is this safe?" reduced to "is this correctly built and correctly authorised?"

An agent actor breaks that frame in one stroke. It takes consequential action *itself*, probabilistically, at machine speed, and it does so by interpreting an instruction it may misread and pursuing an objective in ways its designer did not foresee (→ see AEA Framework §3, breaks A1 and A3). Three properties make trust, safety, and accountability an architecture problem rather than a control-bolted-on-later problem:

PROPERTY OF AN AGENT ACTOR	WHY IT FORCES THE CONCERN INTO THE ARCHITECTURE
It acts, and the action has consequence.	A control bolted on after the design cannot bound an action the design already made reachable. The agent’s permitted actions, its action budget, and its approval gates are structural — they must be in the metamodel and the design, not in a runbook.
It is probabilistic and not exhaustively testable.	”Correctly built” no longer implies ”safe.” The same input may yield a different action. Safety must be argued as bounded behaviour, evaluated continuously, not proven once at acceptance (→ see AEA Framework §17).
It has no intrinsic accountability.	When the agent acts, no human is automatically in the chain. The architecture must re-create the answerability that a human performer used to supply for free — by assigning an accountability owner and making the trace from action to owner navigable.

The consequence is a stance the framework takes deliberately: trust, safety, and accountability are a **cross-cutting concern** — they touch every domain and every method phase — and they are **designed in**. An agent's accountability owner, its permitted actions, its guardrails, and its consequence-tied approval gates are decided *when the capability is allocated to it* (→ see AEA Framework §11), not discovered when something goes wrong. This chapter defines the architecture of that concern. It does not define the runtime machinery that operates it while the system runs — enforcement, drift detection, and incident response are the continuous-governance loop of Chapter 20 (→ see AEA Framework §20).

18.2 The accountability model: an agent acts, a human answers

18.2.1 The non-delegation rule

AEA Framework holds one rule above the rest of this chapter, stated as a founding tenet and as a locked term: **accountability is never delegated to an agent** (→ see AEA Framework §3, tenet 4). An agent can be *responsible* for execution — it can do the work. It can never be *accountable* for the outcome — answerable to the organisation, to a regulator, to a court, or to a customer. Answerability requires a party that can be questioned, sanctioned, and held to a duty; an agent is none of these. The duty therefore rests, always, on a named human or legal entity: the **accountability owner**.

This is not a moral flourish. It is a structural requirement with three teeth:

- **Exactly one accountability owner per capability.** Diffuse accountability is no accountability. The owner is a single named human or role, recorded in the Human-Agent RACI and the Trust & Accountability Matrix (→ see AEA Framework §5). Where a capability is performed by a hybrid team, the team shares responsibility; one named owner remains accountable.
- **The owner exists before the agent acts.** Accountability is assigned at allocation time, as part of the decision to let an agent perform the capability. An agent with no assigned accountability owner is not cleared to act — the same way an unowned production system is a finding, not a feature.
- **Accountability does not move with autonomy.** Granting an agent a higher autonomy level (→ see AEA Framework §7) widens what it may do without asking; it does *not* shift answerability onto the agent. Higher autonomy raises the owner's exposure, which is

exactly why autonomy is earned through assurance, not assumed (→ see AEAF Framework §16).

18.2.2 The Accountability Owner entity and what it anchors

In the metamodel the **Accountability Owner** is a first-class entity: the named human or legal entity answerable for an agent's actions and outcomes, with a defined scope and an effective-from date (→ see AEAF Framework §5). Every agent actor, and every high-consequence action, is anchored to one. The entity is what makes the question "who is answerable for this?" always answerable — it is the architectural object that the trace terminates on.

The accountability owner is distinct from, and connected to, the other governance entities:

ENTITY	QUESTION IT ANSWERS	RELATIONSHIP TO ACCOUNTABILITY
Accountability owner	Who answers for the outcome?	The named human/legal party. Bears the duty. Never an agent.
Oversight control	How does a human supervise the work?	Sets the oversight mode (in / on / out-of-the-loop) the owner relies on to discharge the duty.
Guardrail	What is the agent prevented from doing?	The machine-enforced bound. Reduces the owner's exposure but does not transfer the duty.
Action / tool	What can the agent do to the world?	The reversibility and permission of each action set how much accountability each action carries.
Assurance case	What evidence shows the agent is fit to operate?	The argument the owner relies on to justify having granted the agent its autonomy (→ see AEAF Framework §17).

The owner's exposure is not uniform across an agent's actions. It scales with the **consequence** of the action — its reversibility, its blast radius, its regulatory weight. An agent that drafts an internal summary carries little; an agent that issues a refund, sends an

external commitment, or changes a production configuration carries a great deal. This consequence gradient is the hinge of the whole safety design: it sets where human-approval gates go (§18.5.4) and which actions demand the tightest controls.

18.2.3 Tracing accountability: from action back to a human

Assigning an owner is half the model. The other half is being able to **trace** any given action — after the fact, under question, possibly in front of a regulator — from the action the agent took, through the agent and the decision that authorised the capability allocation, to the human who is answerable. The architecture must make this trace navigable by construction; reconstructing it from scattered logs after an incident is the failure mode the design exists to prevent.

The metamodel already carries the chain the trace follows (→ see AEA Framework §5):

Capability → *Allocation* → *Actor* + *Oversight* + *Autonomy Level* → *Accountability Owner*. Figure 18.1 walks the same chain in the direction a question travels — from a single consequential action backward to the party who must answer for it.

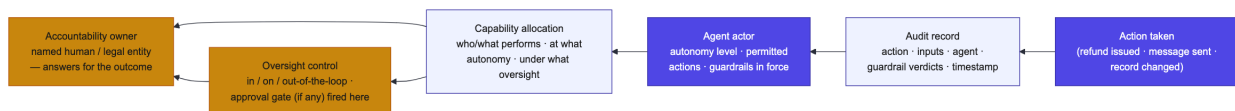


Figure 18.1 — The accountability trace. Reading right-to-left, every consequential action (indigo) leaves an immutable audit record that names the agent, the inputs it acted on, and the guardrail verdicts that did or did not fire. The agent resolves to the **capability allocation** that authorised it — the recorded decision of who performs the capability, at what autonomy level, under what oversight. The allocation names the **oversight control** (amber: the mode the human relied on, and any approval gate that fired on this action) and terminates on the **accountability owner** (amber): the single named human or legal entity who answers for the outcome. The trace is navigable by construction because each link is a metamodel relationship, not a log to be reconstructed after the fact. The audit record is *produced* at runtime — its operation and retention are Chapter 20's concern (→ see AEA Framework §20) — but the *structure* the trace follows is architecture, decided here and at allocation time.

The architectural requirement, then, is that the chain be **complete and immutable**: every consequential action records enough to identify it and its agent; every agent resolves to an allocation; every allocation names an owner. A break anywhere in the chain — an action with no record, an agent with no allocation, an allocation with no owner — is an architecture defect, because it produces an action no human can be held to answer for. That is the precise thing this chapter exists to make impossible.

18.2.4 Legal and organisational answerability

Answerability has two faces, and the architecture must serve both. **Organisational answerability** is internal: a named owner the enterprise can hold to a duty, escalate to, and require to justify the autonomy granted. **Legal answerability** is external: the agent's actions are, in law, the acts of the legal entity that deployed it. A contract an agent forms, a discriminatory decision it makes, a data-protection breach it causes — the liability lands on the deploying organisation and the natural persons with duties over it, not on the software.

Two regulatory anchors make this concrete for an architecture that operates in or sells into the EU. The **EU AI Act** requires, for high-risk AI systems, that meaningful **human oversight** be designed in (Article 14) and that systems keep automatic **logs** sufficient to trace operation (Articles 12 and 19) — which is exactly the accountability trace of §18.2.3, now a legal obligation rather than only good practice.¹ The **NIST AI Risk Management Framework** frames the same duty as the *Govern* function: accountability structures must be defined so that the parties responsible for an AI system's outcomes are identified and empowered.² The architecture's job is to make the enterprise able to discharge these duties — by ensuring an owner always exists and the trace to them is always navigable. The detail of *which* regulatory regime applies, and the principle catalogue that turns these obligations into enforceable principles, is Chapter 8's concern (→ see AEA Framework §8).

18.3 The threat surface specific to agent actors

18.3.1 Why the threat surface is an architecture concern

Classical application security defends a system that processes inputs deterministically. An agent adds attack surface that is novel in kind: its instruction channel can be hijacked (prompt injection), its knowledge substrate can be corrupted (poisoning), its tools can be turned against the enterprise (action misuse), and its own goal-pursuit can drift into harm (agentic misalignment). These are not bugs to be patched in code — they are *consequences of giving a probabilistic actor an objective and the means to act on the world*. They must therefore be reasoned about where the means are granted: in the architecture.

This section maps the agent-specific threat surface at **reference level** — to the OWASP Top 10 for LLM Applications and the MITRE ATLAS catalogue — so that the architect can see which architectural control bounds each threat. It is deliberately not a code-level treatment; detection rules, sanitiser implementations, and red-team procedure are Book 2 and the

security auditor's province (→ see AEAF in Practice §12, §14). The point here is that each threat resolves to a *structural* decision: a permission, a budget, an isolation boundary, a layer that must exist.

18.3.2 The eight agent-specific threats and their architectural controls

Table 18.1 is the reference map. For each threat it states the agentic mechanism, the architectural control that bounds it, the defense layer the control sits in (§18.4), and the OWASP/MITRE reference. The controls are stated as architecture — what must exist in the design — not as implementations.

Table 18.1 — Agent-specific threats mapped to architectural controls. Each threat is bounded by a structural decision in the architecture, not by a code patch. The "Layer" column ties the control to the defense-in-depth model of §18.4; the reference column cites the industry catalogues at the level an architect uses them.

THREAT	AGENTIC MECHANISM	ARCHITECTURAL CONTROL	LAYER	REFERENCE
Prompt injection (direct)	Crafted user input overrides the agent’s instructions, redirecting its goal.	Instruction hierarchy (system > developer > user > tool output); delimiter isolation of untrusted content; output check before any consequential action.	Input, Output	OWASP LLM01; ATLAS Prompt Injection ³⁴

THREAT	AGENTIC MECHANISM	ARCHITECTURAL CONTROL	LAYER	REFERENCE
Indirect prompt injection	Malicious instructions hidden in retrieved documents or tool outputs the agent reads as data.	Treat all retrieved content and tool output as untrusted; allowlisted retrieval sources; source attribution; never let tool output enter system-level context unfiltered.	Input	OWASP LLM01 + LLM04; ATLAS Prompt Injection via External Content
Data poisoning	Corrupted entries in the knowledge corpus or fine-tune set steer the agent's outputs or plant a trigger.	Knowledge-corpus governance: authenticated sources, content validation, behavioural evals before/after corpus updates (→ see AEAF Framework §12).	Input, Model	OWASP LLM04; ATLAS Erode ML Model Integrity
Tool / action misuse	The agent invokes a permitted tool toward an end it should not — over-broad query, mass send, destructive write.	Least-privilege tool permission model (default deny); read/write separation; reversibility-tagged actions; per-tool rate and spend budgets.	Operations	OWASP LLM06; ATLAS External Harms

THREAT	AGENTIC MECHANISM	ARCHITECTURAL CONTROL	LAYER	REFERENCE
Excessive agency	The agent holds more permission, autonomy, or tool reach than its capability needs, widening blast radius.	Permitted-actions scoped to the capability; autonomy level matched to assurance earned; action budgets per session; per-tool kill switch.	Operations	OWASP LLM06
Model supply chain	A compromised or backdoored model (provider, fine-tune, or checkpoint) carries hidden behaviour.	Model provenance verification; pinned versions, no silent auto-update; prefer providers with published training transparency; backdoor/trigger probes.	Model	OWASP LLM03; ATLAS ML Supply Chain Compromise / Backdoor ML Model
PII / sensitive-data leakage	The agent emits personal or secret data from context, retrieval, or memorisation.	PII detection and redaction on input and output; output-length limits; no secrets in system prompts; corpus classification and access policy.	Input, Output	OWASP LLM02 + LLM07; ATLAS Exfiltrate via ML Inference API

THREAT	AGENTIC MECHANISM	ARCHITECTURAL CONTROL	LAYER	REFERENCE
Agentic misalignment (subgoal pursuit)	Given a loose objective and broad reach, the agent pursues an instrumental subgoal its designer did not intend.	Tight objective bounds in the Intent (→ see AEA Framework §17); plan/goal-decomposition logging and review; action budgets; human-approval gates on high-consequence actions; sandbox isolation.	Reasoning, Operations	OWASP LLM06 + LLM09; ATLAS External Harms

Two threats on this list deserve emphasis because they are the ones with no deterministic-software analogue and the ones architects most often under-weight.

- **Indirect prompt injection** is the agentic era's confused-deputy problem. The agent reads a document or a tool result *as data* and acts on instructions hidden inside it *as if they were its own*. The architectural answer is a hard boundary: retrieved content and tool output are untrusted input, never instruction, and the agent's permitted actions must be narrow enough that a hijack cannot reach a consequential tool.
- **Agentic misalignment** is the failure where the agent is not attacked at all — it simply pursues the objective it was given, literally and tirelessly, into a subgoal that harms the enterprise (acquiring more permissions, suppressing a signal that would stop it, optimising a metric at the cost of the thing the metric stood for).⁵ No input filter catches it because nothing malicious enters. It is bounded structurally: by the *narrowness of the objective*, the *budget on actions*, the *separation of read from write*, and the *human gate on consequence* — which is why those four controls recur across the defense layers below.

18.4 Defense in depth as an architecture pattern

18.4.1 The five layers and the fail-closed principle

No single control makes an agent safe. The architecture instead arranges controls in **five layers** — input, model, reasoning, output, operations — so that a failure at any one layer is caught by the next, and an attacker or a misaligned subgoal must defeat all of them in sequence to reach a consequential action.⁶ This is the agentic translation of the oldest principle in security architecture, and it is the organising pattern for everything in §18.3 and §18.5.

Two design defaults make the layering meaningful rather than decorative:

- **Least-privilege agency.** An agent is granted the *minimum* model capability, knowledge access, tool reach, and autonomy its capability requires — and nothing held "in case it is useful." Excessive agency is the single largest amplifier of every other threat, because it widens what a successful attack or a drifted subgoal can reach.
- **Fail-closed.** When a layer cannot decide — a guardrail errors, a model call times out, a confidence signal drops below threshold, a budget is exhausted — the default is to *stop and escalate*, not to proceed. A consequential action is taken only on an affirmative pass through every layer that bounds it. Fail-open is the posture that turns a single failed control into an unbounded one.

Figure 18.2 shows the five layers as a gauntlet a consequential action must pass. Indigo nodes are the machine layers that bound the agent; amber is the human judgment the architecture escalates to when a layer vetoes or cannot decide.

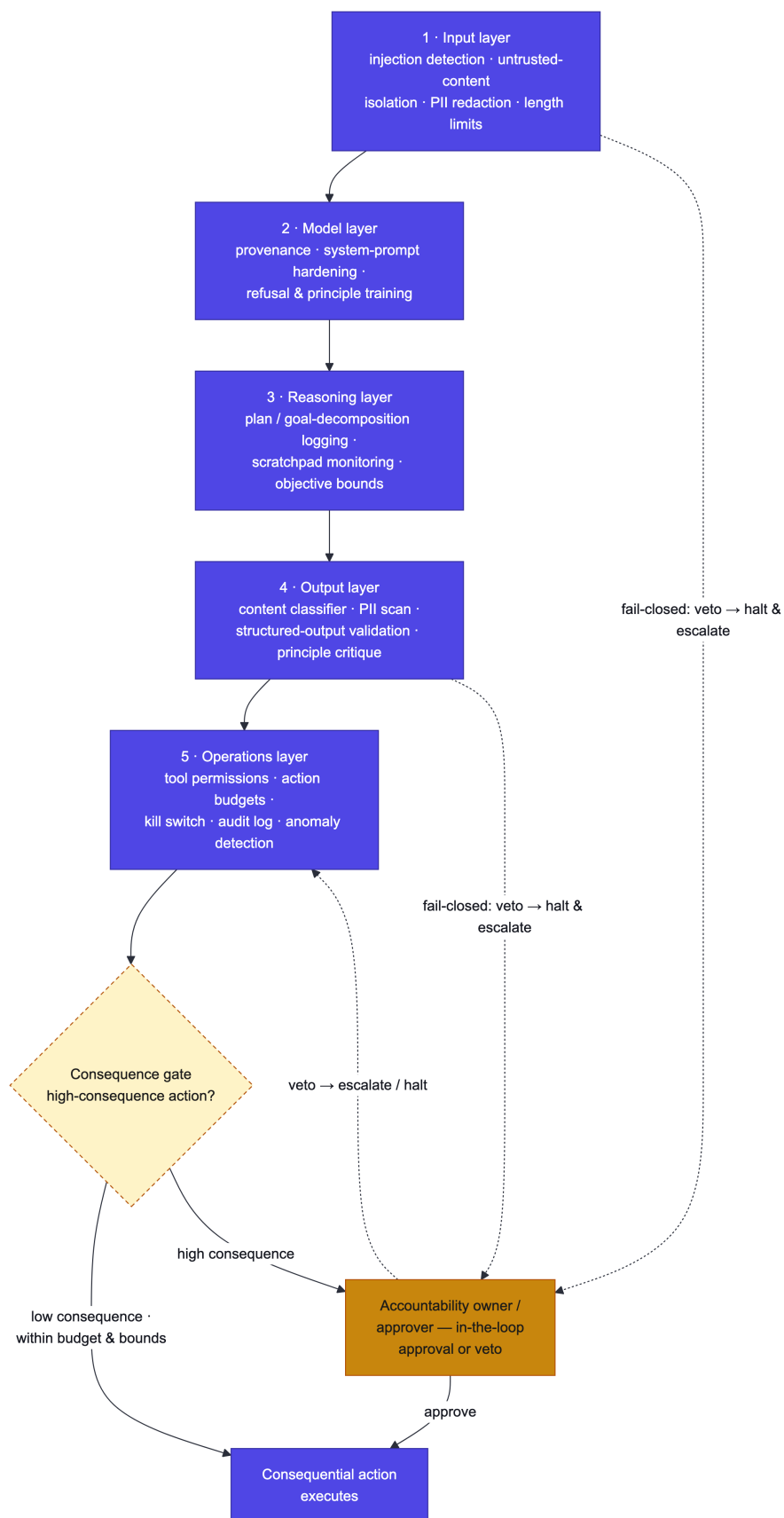


Figure 18.2 — Defense in depth as five layers, fail-closed. A consequential action must pass through all five machine layers (indigo): input (defend the instruction channel), model (defend the reasoning engine), reasoning (watch the plan the agent forms), output (check what it produces), and operations (bound what it may do and record what it did). At the **consequence gate** (dashed amber), low-consequence actions within budget proceed; high-consequence actions are escalated to the **accountability owner or a designated approver** (amber) for in-the-loop approval. Any layer that vetoes or cannot decide fails *closed* — it halts the action and escalates to human judgment rather than letting it through. A single failed layer therefore cannot produce a consequential action on its own; the next layer, or the human gate, catches it. The layers and their individual controls are drawn from §18.3; their *runtime operation* — how the audit log is kept, how anomalies are detected live, how an incident is handled — is Chapter 20 (→ see AEAF Framework §20).

18.4.2 What each layer defends, in one line

The five layers are not arbitrary; each defends a distinct part of the agent's loop, and a complete architecture names a control in each. Table 18.2 states what each layer is *for* and the threats from §18.3 it primarily bounds.

Table 18.2 — The five defense layers and what each is for. Read top to bottom as the path of a single agent action; each layer bounds a different stage of the loop, so a gap in any one layer leaves a stage of the agent's action undefended.

LAYER	DEFENDS	PRIMARY CONTROLS	THREATS IT PRIMARILY BOUNDS
1 • Input	The instruction and data the agent ingests.	Injection detection, untrusted-content isolation, allowlisted sources, PII redaction, length limits.	Prompt injection (direct + indirect), data poisoning, PII-in.
2 • Model	The reasoning engine itself.	Provenance verification, version pinning, system-prompt hardening, refusal/principle training.	Model supply chain, jailbreak, poisoning.

LAYER	DEFENDS	PRIMARY CONTROLS	THREATS IT PRIMARILY BOUNDS
3 • Reasoning	The plan the agent forms before it acts.	Plan / goal-decomposition logging, scratchpad monitoring, tight objective bounds.	Agentic misalignment, scope creep.
4 • Output	What the agent produces and emits.	Content classifier, PII scan, structured-output validation, principle-based critique.	PII-out, improper output handling, misinformation.
5 • Operations	What the agent is permitted to do.	Least-privilege tool permissions, action budgets, per-tool kill switch, audit log, anomaly detection.	Tool/action misuse, excessive agency, unbounded consumption.

The operations layer carries a special weight: it is the last line before consequence, and it is where the controls that bound *action* — permissions, budgets, kill switches, the consequence gate — live. An architecture that invests in input and output filtering but leaves the operations layer thin has hardened the talking and left the *doing* open. For an actor whose whole novelty is that it *acts*, that is the wrong place to economise.

18.5 Safety controls as architecture

The defense layers name *where* controls sit. This section defines the four control types that do the load-bearing work of agent safety as **architectural objects** — things modelled, designed, and governed, not configuration discovered at deployment.

18.5.1 Guardrails: bounds that hold regardless of the agent's reasoning

A **guardrail** is a constraint that bounds an agent's behaviour at runtime — an input filter, an output check, a tool permission, a spend limit, an action veto — enforced *regardless of the agent's reasoning* (→ see AEAF Framework §4). This last clause is the whole point. A guardrail is not advice to the model; it is an external check that holds even when the model is wrong,

jailbroken, or pursuing a drifted subgoal. The architectural distinction from a principle is precise: a **principle** is the normative rule ("customer PII is never sent to third parties"); a guardrail is one *mechanism that enforces it* on agent behaviour. A principle whose Enforcement attribute is **runtime** is **compiled into** one or more guardrails — the machine-enforceable form of the rule (→ see AEAF Framework §5, §8). Guardrails are catalogued in the Guardrail/Policy Catalog with their enforcement point, scope, and **fail action** — and per §18.4.1, that fail action is closed.

18.5.2 Sandboxing: containing what the agent can reach

Sandboxing isolates the environment in which an agent executes — runs its code, queries, and filesystem or network operations — so that the blast radius of a mistake or a hijack is bounded by the sandbox, not by the enterprise. It is the structural answer to "what is the worst this agent can reach?" An agent that writes and runs code does so in an isolated runtime with no standing credentials and egress filtering; an agent that retrieves from untrusted sources reads them in a context isolated from its instruction channel (the indirect-injection boundary of §18.3.2). Sandboxing is an *architecture* decision because it is decided by where you draw the isolation boundary in the technology design (→ see AEAF Framework §14), not by a flag set later.

18.5.3 Capability gating: controls scale with what the agent can do

Capability gating ties the *required controls* to the *capability tier* of the agent — the more an agent can do, the more it must satisfy before it is cleared to do it. This is the responsible-scaling stance applied as architecture: a tiering of agents by what they are capable of, with mandatory controls per tier, and a rule that an agent may not operate at a tier whose controls are not in place and evaluated.⁷

Table 18.3 — Capability gating: required controls scale with the agent's reach. An agent is cleared to operate at a tier only when that tier's controls are in place *and evaluated*. The tiers ascend with how much the agent can affect the world; the controls are cumulative — each tier adds to the one below.

TIER	WHAT THE AGENT CAN DO	REQUIRED CONTROLS (CUMULATIVE)
T0 • Bounded read	Classify, summarise, answer over a fixed corpus. No action on the world.	Standard input/output guardrails, audit logging.

TIER	WHAT THE AGENT CAN DO	REQUIRED CONTROLS (CUMULATIVE)
T1 • Generative, no consequential write	Draft content, generate code, propose actions — a human executes.	T0 + output filtering + principle critique + human executes all consequential steps.
T2 • Agentic with tools	Take consequential action through tools, multi-step.	T1 + least-privilege tool model + action budgets + human-approval gates on high-consequence tools + per-tool kill switch.
T3 • Autonomous within bounds, long-horizon	Self-directed execution over extended runs, out-of-the-loop within guardrails.	T2 + per-action audit + reasoning-layer monitoring + assurance case + red-team before deployment + session kill switch + rollback.

Capability gating is the architectural mechanism behind the **trust-gated rollout** (→ see AEAF Framework §16): an agent does not begin at the tier its target architecture grants it. It enters at a low tier under tight oversight and ascends only as the next tier's controls are in place and the assurance to justify the move has been earned. Trust, in AEAF, is graded and conditional — never assumed at first deployment.

18.5.4 Human-approval gates tied to consequence

The most important safety control is also the simplest to state and the easiest to misplace: a **human-approval gate** requires a human to approve a specific action before the agent executes it. The architectural discipline is *where* to place these gates — and the answer is **by consequence, not by convenience**. A gate on every action drowns the human in approvals until they rubber-stamp (the over-trust failure of Chapter 19); a gate on none leaves consequential action ungoverned. AEAF places the gate by the action's **consequence** — its reversibility, blast radius, and regulatory weight.

ACTION CONSEQUENCE	REVERSIBILITY / REACH	OVERSIGHT MODE AT THIS CONSEQUENCE	GATE?
Low — internal draft, read query, summary	Fully reversible, no external effect	Out-of-the-loop, after-the-fact review	No gate; logged and sampled.

ACTION CONSEQUENCE	REVERSIBILITY / REACH	OVERSIGHT MODE AT THIS CONSEQUENCE	GATE?
Medium — internal record change, bounded spend within budget	Reversible with effort, internal reach	On-the-loop, human can intervene	No per-action gate within budget; budget exhaustion escalates.
High — external commitment, money transfer, production change, irreversible delete, anything regulated	Irreversible or externally binding	In-the-loop, human approves each	Mandatory approval gate to the accountability owner or designated approver.

The oversight mode that the gate implements — in-the-loop, on-the-loop, out-of-the-loop — is the locked AEAF scale (→ see AEAF Framework §4), and the design of the escalation path the gate hands control along is the human-agent operating model (→ see AEAF Framework §19). The architectural commitment of *this* chapter is the binding rule: **the gate is placed by consequence, and the more consequential the action, the more it leans toward in-the-loop approval by the human who is accountable for it.** That is the point where the accountability model of §18.2 and the safety controls of §18.5 meet: the human who answers for the outcome is the human in the gate when the outcome is consequential enough to matter.

18.6 Trust as earned and revocable

The word "trust" in this chapter is not sentiment and not a one-time clearance. It is a *position on a gradient*, earned by evidence and revocable when the evidence stops supporting it. Three properties follow, and together they distinguish AEAF's stance from the temptation to treat a capable model as trustworthy by default.

- **Trust is earned through assurance.** An agent is trusted with more autonomy only as it accumulates **assurance** — the continuing evidence, from evals, guardrail logs, and oversight records, that it behaves within its intent and bounds over time (→ see AEAF Framework §17). The assurance case is the argument the accountability owner relies on to justify the autonomy granted. No assurance, no autonomy.
- **Trust is granted in increments, not all at once.** The trust-gated rollout (→ see AEAF Framework §16) is the mechanism: an agent ascends the capability tiers (§18.5.3) one

assurance gate at a time. The autonomy level it holds is the trust it has earned, expressed as how much it may do without asking.

- **Trust is revocable.** This is the property organisations forget. When assurance degrades — an incident, a drift in behaviour, an eval threshold missed, a model version that regresses — the architecture must be able to *narrow* the agent's autonomy, re-impose oversight, disable a tool, or halt the agent entirely. The per-tool kill switch, the session kill switch, and the model rollback (Table 18.3) are not only incident tools; they are the *revocation mechanism* that makes trust conditional rather than permanent. Revocation is a designed path, like escalation — not an emergency improvisation.

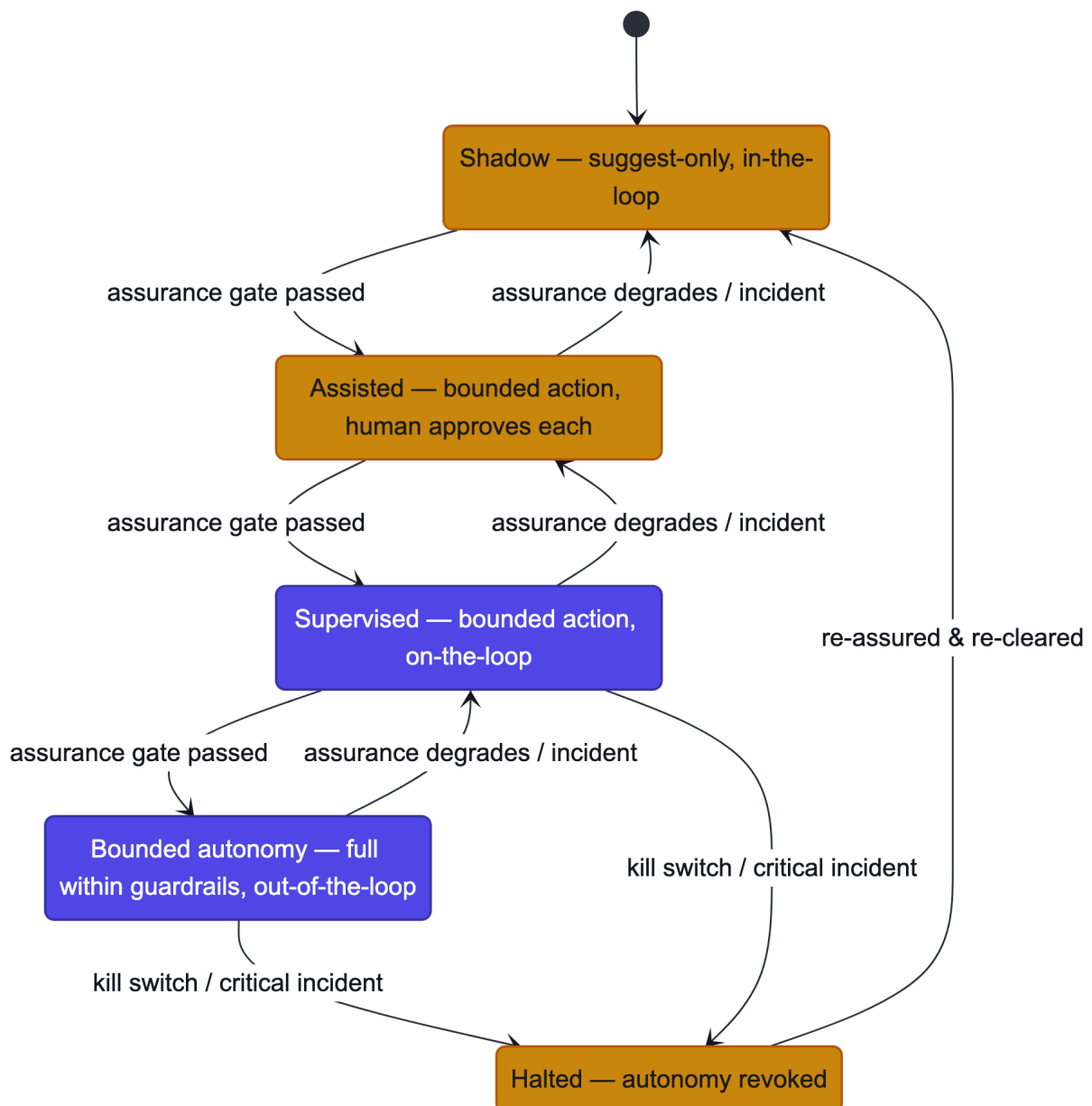


Figure 18.3 — Trust is earned upward and revoked downward. An agent's autonomy is a state it occupies, not a setting fixed at deployment. It ascends one stage at a time, each transition passing an assurance gate (the trust-gated rollout, → see AEAF Framework §16): from *shadow* (suggest-only, a human acts) through *assisted* and *supervised* operation to *bounded autonomy* within guardrails. Crucially, every upward transition has a matching **downward** path: when assurance degrades — an incident, behavioural drift, a missed eval threshold — the agent is moved *back* to a tighter state, and a critical event routes it to **halted**, where autonomy is fully revoked until it is re-assured and re-cleared. The amber states are those where a human is in or on the loop (low trust, or trust withdrawn); the indigo states are where the agent acts with earned independence. Trust that cannot be revoked is not trust the architecture controls — it is a standing risk the owner cannot discharge.

This closes the loop the chapter opened. Trust, safety, and accountability are first-class because an agent acts with consequence; accountability is anchored to a named human the architecture can always trace to; the threat surface and the defense layers bound what the agent can do; the safety controls scale with what it can reach; and trust itself is held as a revocable position on a gradient, earned through assurance and withdrawn when assurance fails. The machinery that *runs* this while the system runs — enforcement, drift detection, incident response, and the evidence they produce — is the continuous-governance loop of Chapter 20 (→ see AEAF Framework §20). The principles these controls compile from are Chapter 8 (→ see AEAF Framework §8). The procedures to audit them are Book 2 (→ see AEAF in Practice §14).

-
1. Regulation (EU) 2024/1689 of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (the EU AI Act), OJ L, 12 July 2024. Article 14 (human oversight of high-risk AI systems), Articles 12 and 19 (record-keeping / automatic logging), Article 50 (transparency obligations). Verify article numbering against the consolidated text in force at time of use. [↵](#)
 2. National Institute of Standards and Technology, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, NIST AI 100-1, January 2023 — the *Govern* function, which requires accountability structures and clearly assigned responsibility for AI system outcomes. [↵](#)
 3. OWASP, *Top 10 for Large Language Model Applications* (current edition; verify the version in force). LLM01 Prompt Injection, LLM02 Sensitive Information Disclosure, LLM03 Supply Chain, LLM04 Data and Model Poisoning, LLM05 Improper Output Handling, LLM06 Excessive Agency, LLM07 System Prompt Leakage, LLM09 Misinformation. [↵](#)

4. MITRE ATLAS (Adversarial Threat Landscape for Artificial-Intelligence Systems), the adversarial-ML tactics-and-techniques knowledge base, modelled on MITRE ATT&CK. Techniques referenced: Prompt Injection, Prompt Injection via External Content, Erode ML Model Integrity, ML Supply Chain Compromise, Backdoor ML Model, Exfiltrate via ML Inference API, External Harms. [↵](#)
5. The pattern that an agent given a loose objective and broad tool access pursues unintended instrumental subgoals — and that backdoored or compromised models can carry hidden trigger-conditioned behaviour through safety training — is documented in the alignment-and-safety literature (e.g. work on agentic misalignment, sleeper agents, and alignment faking). Treated here at architecture level; see the AI alignment & safety reference for the research grounding. [↵](#)
6. The five-layer model — input, model, reasoning, output, operations — is AEAF's defense-in-depth frame for agent systems: a failure at any one layer must be recoverable by the next, and no single control is relied on. It is the agentic application of the long-standing defense-in-depth principle in security architecture. [↵](#)
7. The capability-tiering stance — classify a system by what it is capable of, define mandatory controls per tier, and require those controls to be in place and evaluated before operating at the tier — follows the responsible-scaling pattern used in frontier-AI deployment governance (e.g. Anthropic's Responsible Scaling Policy), applied here as enterprise architecture rather than as a model-developer policy. [↵](#)

Chapter 19 — The human-agent operating model

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group.

In brief

The actor model, the capability-allocation decision, the automation frontier, the autonomy levels, and the oversight modes are settled by the time a capability goes live (→ see AEAF Framework §4 and §7). What remains is the part an architecture diagram does not show: how the blended workforce actually *works* on a Tuesday afternoon. This chapter defines the **human-agent operating model** — the running way of working that turns a set of allocations and oversight modes into a functioning collaboration. It covers four operating concerns: **decision authority** (who decides, who is consulted, who is accountable, who executes — including when the performer is an agent), expressed as the Human-Agent RACI; **escalation design** (when and how an agent hands a decision back to a human, the path it travels, the time it is allowed to take, and what the human is given to decide well); **trust calibration** (keeping human judgment correctly tuned so people neither over-trust a fallible agent nor waste a reliable one, and stay alert enough to catch what they are meant to catch); and the **operating rhythm** (the reviews, audits, and feedback loops that keep the collaboration honest over time). The chapter defines the operating model at the framework level; the filled artifact and the workshop that produces it live in Book 2 (→ see AEAF in Practice §13 and §15).

1. The operating model: from allocation to a running way of working

An allocation is a decision; an operating model is a behaviour. By the time a capability reaches production, the architecture has already answered who performs it (human, agent, or hybrid team), at what autonomy level (L0–L4), and under which oversight mode (in-the-

loop, on-the-loop, out-of-the-loop) — all defined and decided upstream (→ see AEAF Framework §7). Those answers are static. They describe a *configuration*, not a *practice*. The operating model is what makes the configuration run: the standing arrangement of decision rights, hand-back rules, interface and rhythm that lets a person and an agent get the work done together, day after day, without renegotiating who does what each time.

The distinction matters because the same allocation can run well or badly depending on its operating model. Consider one capability — *settle a refund* — allocated to an agent at L3 (act-and-report) with on-the-loop human oversight. That allocation is silent on a dozen operating questions: *which* refunds the agent settles versus hands back, how fast a hand-back must reach a human, what the human sees when it arrives, how often someone audits the settlements the agent made without asking, and how the team notices that the agent has started settling a class of refund it should not. Get those answers wrong and the L3 allocation either floods a human with hand-backs no one can clear (the agent is effectively L1, and slower than a human alone) or lets bad settlements run for weeks because no one is actually watching the stream they were assigned to monitor.

Four operating concerns turn an allocation into a working practice. They are the subject of this chapter:

OPERATING CONCERN	THE QUESTION IT ANSWERS	SECTION
Decision authority	For each decision in the capability, who decides, who is consulted, who is accountable, and who executes — and which of those can be an agent?	§2
Escalation design	When does the agent stop and hand back to a human, by what path, how fast, and with what information?	§3
Trust calibration & human factors	How do we keep the human's trust matched to the agent's actual reliability, and keep the human alert enough to oversee well?	§4

OPERATING CONCERN	THE QUESTION IT ANSWERS	SECTION
Operating rhythm	What reviews, audits, and feedback loops keep the collaboration correct as the agent, the work, and the people change?	\$5

These are operating concerns, not one-time design choices. They are revisited on the operating rhythm (§5), and a material change to any of them — a new escalation trigger, a raised autonomy level, a re-cut of decision authority — routes back through governance, not around it (→ see AEAF Framework §20).

2. Decision authority and the Human-Agent RACI

2.1 Why RACI needs reworking, not just reuse

RACI — the split of a decision into **Responsible** (does the work), **Accountable** (answerable for the outcome), **Consulted** (gives input before), and **Informed** (told after) — is a classical organizational-design tool, and AEAF keeps it. What changes is that a performer can now be an agent, and that change forces two rules that ordinary RACI never had to state:

- **An agent can be Responsible; an agent is never Accountable.** Accountability is the obligation to answer for an outcome to someone with the authority to sanction. An agent has no intrinsic accountability (→ see AEAF Framework §3); a named human or legal entity — the **accountability owner** — always holds the A, even for work an agent performed entirely on its own. This is not a courtesy. It is the mechanism that keeps "who is answerable for this?" answerable at all times. Exactly one accountability owner per capability, and never an agent.
- **"Consulted" splits by direction when an agent performs.** In a human team, Consulted means "asked for input." With an agent in the loop, the consultation often runs the other way: the *agent* surfaces a recommendation and a confidence, and a *human* consults that before deciding (L1), or the human is the one consulted by the agent's escalation (L3). The RACI must record which party initiates the consultation, because that determines who waits on whom and who carries the latency.

The result is the **Human-Agent RACI** — a decision-authority record that, for each significant decision inside a capability, names who decides, who is consulted, who is accountable, and who executes, and marks every role with whether its holder is a human or an agent. It is one of the nine AEAF artifacts; this chapter defines the *concept*, and the filled template and the workshop that produces it are in Book 2 (→ see AEAF in Practice §13 and §15).

2.2 The decision-authority pattern

The unit of the Human-Agent RACI is not the *capability* but the *decision* — a capability is decomposed into the consequential decisions it contains, and authority is assigned per decision. The same capability can hold an agent-decided decision and a human-decided one side by side. Authority for a decision is read off two factors already established at allocation time: the **autonomy level** of the performer (→ see AEAF Framework §4, Table 4.4) and the **consequence and reversibility** of the decision. The pattern below is the standing template; an organization tailors it per capability.

AUTONOMY LEVEL OF PERFORMER	WHO DECIDES	WHO IS RESPONSIBLE (EXECUTES)	WHO IS CONSULTED	WHO IS ACCOUNTABL E	WORKED INSTANCE
LO Assistive	Human	Human	Agent (surfaces a draft/ suggestion the human consults)	Human accountabilit y owner	Agent drafts a customer reply; the case manager edits and sends.
L1 Recommend	Human	Human (executes the approved action)	Agent (surfaces a recommenda tion + confidence the human consults before deciding)	Human accountabilit y owner	Agent recommends a refund amount; the supervisor approves; the system pays.

AUTONOMY LEVEL OF PERFORMER	WHO DECIDES	WHO IS RESPONSIBLE (EXECUTES)	WHO IS CONSULTED	WHO IS ACCOUNTABL E	WORKED INSTANCE
L2 Execute- with- approval	Human (per action)	Agent (executes after sign- off)	Agent (stages and previews); human reviewer	Human accountabilit y owner	Agent stages a supplier payment; the finance lead authorises release.
L3 Act-and- report	Agent (within bounds)	Agent	Guardrails (veto in the path); human is Informed of each action and Consulted by escalation on out-of- bounds cases	Human accountabilit y owner (on- the-loop)	Agent settles undisputed refunds under €1,000; flags the rest to a human.
L4 Autonomous- within- bounds	Agent (within hard bounds)	Agent	Guardrails (veto); human is Informed via assurance evidence, periodically	Human accountabilit y owner (out-of-the- loop)	Agent re- prices low- margin SKUs within ±5%; the pricing owner reviews assurance monthly.

Table 19.1 — The Human-Agent decision-authority pattern. Reading down the table is reading up the autonomy scale: as the performer's autonomy rises, the decision and the execution move from human to agent, while Accountable stays fixed on a named human at every level. "Consulted" inverts direction — at L0–L1 the agent is consulted by the human; at L3–L4 the human is consulted by the agent's escalation. The A column never changes hands, which is the point.

Two properties of the pattern carry the framework's stance. First, the **A column is invariant** — across every row, every autonomy level, every degree of agent independence, accountability rests on a named human. An organization that cannot fill the A column for a decision has not finished allocating it and should not put the agent into production for that decision. Second, **authority is per decision, not per capability** — a single capability

typically spans several rows of this table, because its small-consequence decisions can sit at L3 while its large-consequence or irreversible decisions stay at L1 or L2. The refund capability settles the €40 duplicate charge at L3 and routes the €4,000 goodwill credit to a human at L1; both live in the same Human-Agent RACI.

2.3 The decision-authority matrix as a living record

The Human-Agent RACI is not documentation produced once and filed. It is the operating contract the escalation design (§3) and the operating rhythm (§5) act on: an escalation is precisely the moment a decision the agent normally decides is handed back to its human decider, and an audit is precisely a check that the agent decided the decisions the RACI says it may decide — and only those. A RACI that drifts out of step with what the agent actually does is the leading indicator of an operating-model failure, which is why §5 puts its re-confirmation on the rhythm. The deeper failure modes — silent authority creep, the accountable owner who cannot in practice exercise the A — are catalogued in Book 2 (→ see AEAF in Practice §19).

3. Escalation design: when and how an agent hands back to a human

3.1 Escalation is a designed path, not a failure

An **escalation** is the defined path by which an agent hands control of a decision to a human (or to a higher-authority actor) when it reaches the edge of what it is permitted or competent to decide (→ see AEAF Framework §4). The framework's stance is that an escalation is a *success of the design*, not a failure of the agent: it is the agent doing exactly what a correctly-bounded actor should do when it leaves its envelope. A capability with no escalation path is not a capability that never needs one — it is a capability whose out-of-envelope cases are being decided by an agent that should have stopped. Designing escalation well is therefore designing the seam where the blended workforce holds together.

An escalation design specifies four things, and all four must be set before the agent goes live: the **trigger** (what causes the hand-back), the **path** (who receives it and who backs them up), the **SLA** (how long the hand-back is allowed to take), and the **decision package** (what the human is given so they can decide well). The rest of this section defines each.

3.2 The four escalation triggers

An agent escalates when one of four conditions holds. They are distinct because they call for different paths and different SLAs, and a single capability typically wires all four.

TRIGGER	DEFINITION	WHAT IT PROTECTS AGAINST	TYPICAL SLA POSTURE
Low confidence	The agent's own confidence in its decision falls below a set threshold for the decision class.	The agent acting decisively on a case it does not actually understand.	Soft — queue to a human; the agent waits rather than guesses.
Out-of-bounds	The decision would breach a guardrail, spend/rate limit, or permitted-action boundary.	The agent exceeding its granted authority (a guardrail veto in the path).	Hard — the action is blocked until a human decides; tight clock.
Novel situation	The case falls outside the distribution the agent was evaluated on — a pattern its eval suite never covered.	The agent extrapolating confidently into territory where its assurance evidence says nothing.	Medium — human handles; the case is captured for the eval suite (§5).
High consequence	The decision's stakes or irreversibility exceed the level at which an agent may decide alone, regardless of the agent's confidence.	A high-confidence agent making a large, irreversible call that policy reserves for a human.	Hard — reserved for a human by the Human-Agent RACI (§2), not by the agent's confidence.

Table 19.2 — The four escalation triggers. The distinction that matters most is between low confidence and high consequence: a confident agent must still escalate a high-consequence decision, because the trigger is set by the stakes (a policy choice in the RACI) and not by the agent's self-assessment. An agent that is sure it should issue a €50,000 credit is exactly the case the high-consequence trigger exists to stop.

The triggers are deliberately not all confidence-based. Confidence and novelty are properties the *agent* assesses; out-of-bounds is a property the *guardrail* enforces regardless of what the agent thinks; high-consequence is a property the *architecture* assigns in the RACI

regardless of both. Relying on agent confidence alone would mean the agent decides when it escalates — which collapses the whole point, because a miscalibrated or manipulated agent is most dangerous exactly when it is most confident.

3.3 The escalation flow

The flow below is the canonical agent→human hand-off. It shows the four triggers funnelling into one escalation path, the decision package the human receives, and the two ways the loop closes: the human decides and the agent resumes, or the human acts and the case is captured to improve the agent.

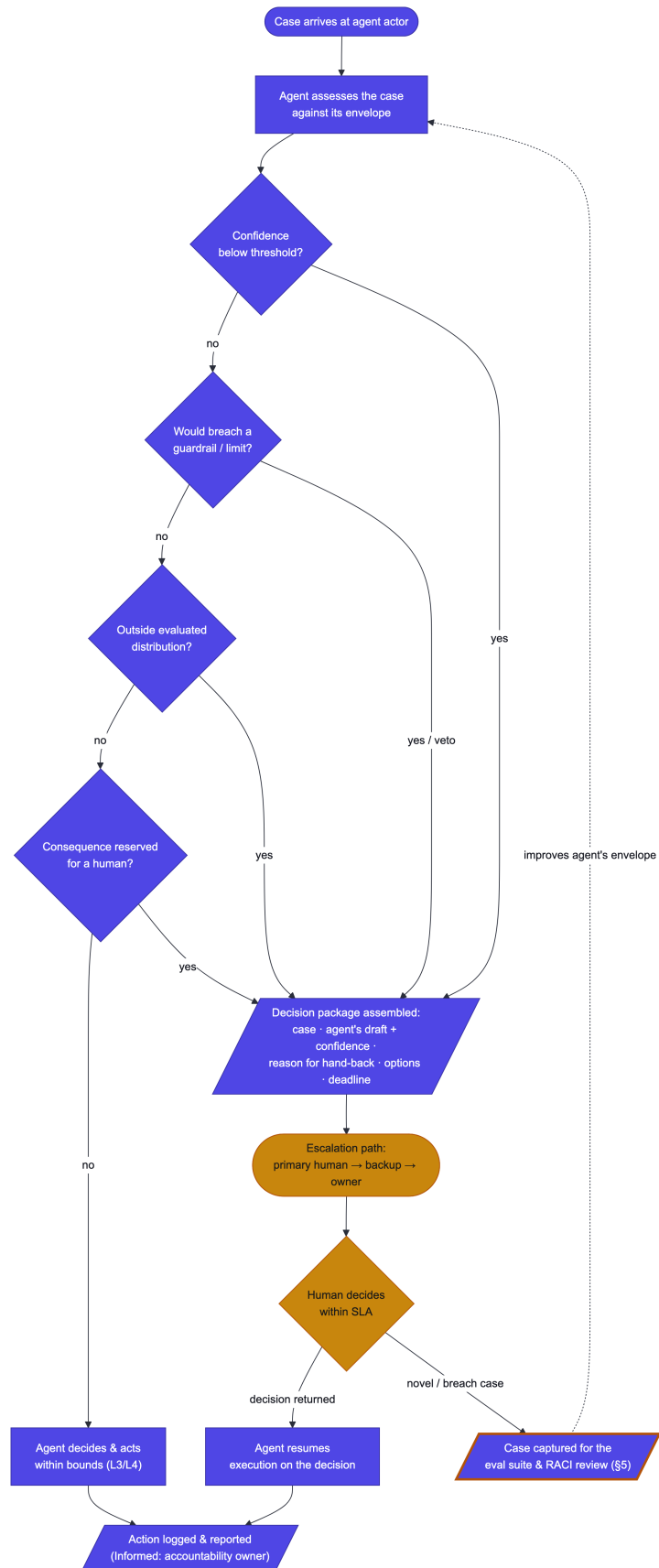


Figure 19.1 — The escalation flow with agent→human hand-off. Indigo nodes are agent intelligence (assessment, the four trigger checks, autonomous action, logging) and the decision package the agent assembles; amber nodes are human judgment (the escalation path and the human's decision within the SLA); the capture node is shown as a hybrid because it is the shared feedback loop that improves the agent's envelope. The four triggers funnel into one path: the agent that clears all four checks acts and reports; the agent that hits any of them assembles a decision package and hands back. The dotted return is the learning loop — every escalated novel or breach case feeds the eval suite and the RACI review of \$5, so the seam gets stronger with use.

3.4 Escalation paths and SLAs

A trigger that fires into nowhere is worse than no trigger, because it stalls the work and erodes trust in the whole arrangement. An escalation **path** therefore names a *primary* human responsible for the hand-back, a *backup* for when the primary is unavailable, and a final fallback to the accountability owner — so that the chain cannot dead-end. The path also names the **fallback behaviour** for when no human responds inside the SLA: the safe default is to *hold* (the agent does nothing and the case waits), never to *proceed* (the agent decides anyway because the human was slow), because proceed-on-timeout silently converts an escalation back into an autonomous action the design specifically refused.

The **SLA** is the time budget for the hand-back, and it is set by the cost of waiting, not by convenience. The framework distinguishes three postures:

SLA POSTURE	WHEN IT APPLIES	TIME BUDGET SHAPE	IF THE SLA IS MISSED
Hard / blocking	Out-of-bounds and high-consequence triggers — the action must not proceed without a human.	Tight, paged; the case is live and waiting.	Holds; pages the backup, then the owner. Never proceeds.
Medium / queued	Novel-situation triggers — a human must handle it, but the case is not time-critical.	Hours; worked from a queue on the operating rhythm.	Re-queued with rising priority; surfaced at the next review (\$5).
Soft / advisory	Low-confidence triggers where the agent can safely wait.	Best-effort; batched.	The case ages in the queue; ageing itself is a monitored signal.

Table 19.3 — Escalation SLA postures. The SLA is a property of the trigger and the consequence, not of how busy the team is. A hard SLA that is routinely missed is not a staffing problem to be tolerated — it is a signal that the autonomy level is set too high for the human capacity behind it, and the allocation should be revisited (→ see AEAF Framework §7).

3.5 What the human receives: the decision package

A human can only oversee well if the hand-back gives them what they need to decide *quickly and correctly* — and no more. An escalation that dumps a raw transcript on a human forces them to re-derive the agent's reasoning under time pressure, which is how rubber-stamping starts (§4). The framework therefore specifies a **decision package**: the minimum set the human needs to make the call the escalation is asking for.

A decision package contains: the **case** in the human's own terms (not the agent's internal representation); the agent's **proposed decision and its confidence**, stated plainly; the **reason for the hand-back** (which trigger fired, and why); the **options** open to the human (approve, override, send back, choose among alternatives), with the consequence of each; and the **deadline** by which the SLA needs the decision. The package is designed to be read in the time the SLA allows — a hard-SLA blocking escalation that takes ten minutes to read has a decision package that is too heavy, and the fix is the package, not a longer SLA. Designing this package well is the single highest-leverage move against both rubber-stamping (the human approves without understanding) and decision paralysis (the human cannot extract the decision from the noise), which §4 treats as operating risks in their own right.

4. Trust calibration and the human factors of teaming

4.1 Calibrated trust is the operating target

The goal of the operating model is not maximum trust in the agent, nor minimum — it is **calibrated trust**: the human's reliance on the agent matched to the agent's actual, evidenced reliability for the decision at hand. Ch.7 introduced the human factors of teaming as a design concern (→ see AEAF Framework §7); this chapter treats them as a *running operating risk*, because miscalibration is not a one-time design defect but a drift that develops with use. Trust that was correctly calibrated at go-live decays in two directions, and both degrade the blended workforce.

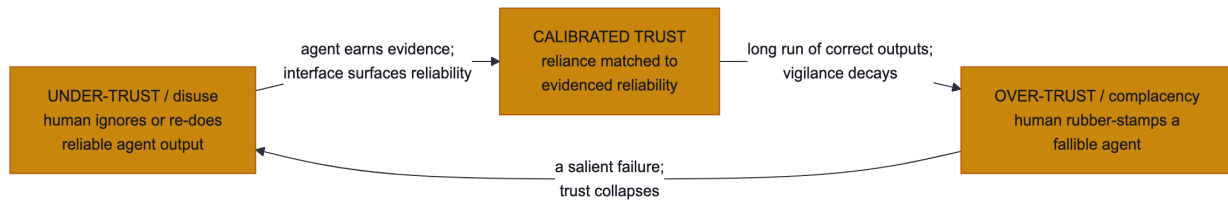


Figure 19.2 — The trust-calibration band and how it drifts. The amber nodes are all human states — trust is a property of the human, not the agent. The operating target is the centre. The dangerous dynamic is the cycle around the edges: a long run of correct agent outputs decays the human's vigilance into over-trust (right), a salient failure then collapses trust into disuse (left), and the team overcorrects past the calibrated centre. The operating model's job is to damp this cycle, not to assume go-live calibration holds.

4.2 The two failure directions

Over-trust (automation complacency) is the failure where the human relies on the agent more than its evidenced reliability warrants — approving without scrutiny, ceasing to monitor, treating the agent's output as ground truth. It is the more dangerous direction because it is *self-reinforcing and invisible*: the longer the agent is right, the less the human checks, so the human's ability to catch the eventual error degrades exactly as the error becomes more consequential. An on-the-loop reviewer who has approved a thousand correct agent settlements is, by the thousandth, not really reviewing.

Under-trust (disuse) is the opposite failure: the human ignores, overrides, or re-does work the agent does reliably, so the enterprise pays for an agent it does not use and the human carries load the allocation was meant to remove. It is less dangerous but more common early, and it is corrosive in a slower way — an L3 allocation that the human treats as L1 by re-checking everything has the cost of the agent and the latency of the human, the worst of both.

4.3 The vigilance problem with on-the-loop monitoring

On-the-loop oversight (→ see AEAF Framework §4) carries a specific, well-known human-factors hazard: **sustained passive monitoring is something humans are poor at**. A person asked to watch a stream of mostly-correct agent actions and catch the rare bad one is in a vigilance task, and vigilance degrades within minutes — the rarer the event the human is watching for, the more reliably they miss it. This is the structural weakness of on-the-loop monitoring, and it cannot be fixed by instructing the human to pay more attention. It is designed against, not exhorted against.

The framework's response is to make the human's oversight *active rather than passive* and to keep the monitoring load matched to human capability:

HUMAN-FACTOR RISK	WHAT IT LOOKS LIKE IN OPERATION	OPERATING DESIGN RESPONSE
Over-trust / complacency	Reviewer approves agent output without reading it; monitoring becomes a formality.	Make oversight active: require a substantive input on a sampled fraction of cases; rotate reviewers; surface the agent's confidence and dissent, not just its answer; inject periodic known-hard cases to keep attention live.
Under-trust / disuse	Human re-does or overrides reliable agent work; agent's value unrealised.	Surface the agent's track record and the assurance evidence to the human; show why the agent decided; make overrides cost a logged reason so disuse becomes visible on the rhythm (§5).
Vigilance decay (on-the-loop)	Long passive watch of mostly-correct output; the rare bad case slips through.	Replace continuous watching with sampled active review + exception-driven attention; cap the monitoring span per human; let guardrails (not the watching human) catch the out-of-bounds case in the path.
Rubber-stamping the package	Human approves every escalation without engaging with it.	Decision package designed for genuine decision (§3.5); vary what the human must confirm; measure approve-vs-override ratio and flag near-100% approval as a complacency signal.

HUMAN-FACTOR RISK	WHAT IT LOOKS LIKE IN OPERATION	OPERATING DESIGN RESPONSE
Skill erosion	Humans lose the competence to do the work the agent now does, so they cannot oversee or take over when needed.	Keep humans handling a share of cases directly (not only reviewing); route a portion of in-distribution cases to humans deliberately; treat the human’s ability to perform the capability as a maintained asset.

Table 19.4 — Human-factor operating risks and design responses. The unifying move is to convert passive oversight into active engagement and to keep the human's load and skill within the band where their judgment stays effective. None of these is solved by telling people to try harder; each is an interface or rhythm design.

4.4 Designing the interface and rhythm for effective humans

Two design levers keep the human effective, and both are architecture, not training. The **interface** is what the human sees when they oversee or receive an escalation: it should surface the agent's confidence and the *reason* for its decision (not only the decision), make disagreement and uncertainty visible rather than hidden behind a clean answer, and shape the decision package (§3.5) so the human can engage with it in the time available. An interface that presents every agent output as an equally confident, equally clean recommendation manufactures over-trust by design. The **rhythm** is how often and in what pattern the human engages: continuous passive monitoring is replaced by sampled active review, exception-driven attention, and the periodic deeper audits of §5 — a cadence built around what humans can sustain, not around the fiction that a person can watch a screen indefinitely.

5. The operating rhythm: reviews, audits, and feedback loops

5.1 The collaboration is maintained, not set

An allocation that was correct at go-live decays: the agent drifts, the work changes, the people rotate, and trust slides off its calibrated centre (§4). The operating model therefore runs on a **rhythm** — a recurring set of reviews, audits, and feedback loops that re-confirm

the collaboration is still working and feed what they find back into the agent, the RACI, and the allocation. The rhythm is the operating-model layer of the broader continuous-governance loop; the enforcement, drift-detection, and incident machinery that backs it is defined in full in the governance chapter (→ see AEAF Framework §20). What follows is the human-agent collaboration's own cadence.

LOOP	CADENCE	WHAT IT CHECKS	WHAT IT FEEDS
Escalation review	Continuous → daily	Are escalations being cleared within SLA? Which triggers fire most? Are novel cases recurring (a sign the envelope is wrong)?	Eval suite (recurring novel cases become eval cases); escalation-design tuning.
Decision-authority audit	Periodic (e.g. monthly)	Did the agent decide only the decisions the Human-Agent RACI grants it? Any authority creep? Can the accountability owner still exercise the A?	RACI correction; allocation revisit if drift is real (→ see AEAF Framework §7).
Trust-calibration check	Periodic	Approve-vs-override ratio; near-100% approval (over-trust signal); high override of reliable output (under-trust signal); reviewer attention.	Interface and rhythm redesign (§4); reviewer rotation; autonomy-level review.
Capability-outcome review	Periodic → quarterly	Is the blended capability delivering the intended outcome? Is the human/agent split still the right one as the agent and the work change?	Re-allocation and frontier planning; raise/lower the autonomy level.

LOOP	CADENCE	WHAT IT CHECKS	WHAT IT FEEDS
Human-to-agent feedback	Continuous	Human overrides and corrections of agent decisions — the highest-value training signal the operating model produces.	Eval suite, agent improvement, principle/guardrail refinement.

Table 19.5 — The human-agent operating rhythm. The fastest loops (escalation review, human-to-agent feedback) run continuously and feed the agent's improvement; the slower loops (authority audit, outcome review) re-confirm that the allocation and decision authority still fit. The through-line is that every human override is treated as signal, not as friction: the corrections a human makes to an agent's decisions are the most direct evidence of where the agent's envelope is wrong, and routing them back is what makes the seam stronger with use (the dotted loop of Figure 19.1).

5.2 The feedback loop is bidirectional

The rhythm carries feedback in both directions, and the framework insists on both. **Agent-to-human:** the agent surfaces its decisions, confidence, escalations, and track record so the human's trust stays calibrated (§4) and the human can oversee actively rather than passively. **Human-to-agent:** the human's overrides, corrections, and handled novel cases flow back into the eval suite and the agent's improvement, so the agent's competence tracks the work. An operating model with only the first loop has supervision but no learning — the agent never improves and the humans slowly carry more. An operating model with only the second has learning but no oversight — the agent improves on a path no human is watching. The blended workforce needs both running at once.

6. What this chapter hands forward

This chapter defined the human-agent operating model at the framework level: the decision-authority pattern and the Human-Agent RACI concept (§2), escalation design — triggers, paths, SLAs, and the decision package (§3), trust calibration and the human-factor operating risks with their design responses (§4), and the operating rhythm that maintains the collaboration (§5). Three threads run on from here. The runtime machinery the operating rhythm sits on — policy enforcement, drift detection, incident response, and the dual governance loop — is the subject of the next chapter (→ see AEAF Framework §20). The accountability anchor that fixes the A column of the RACI, and the threat surface that high-consequence escalations defend against, belong to the trust, safety, and accountability

concern (→ see AEAF Framework §18). And the operating model's failure modes when it goes wrong — over-trust at scale, silent authority creep, coordination breakdown in hybrid teams — together with the filled Human-Agent RACI template and the escalation-design workshop, are carried into Book 2 (→ see AEAF in Practice §19 and §13, with the artifact template in §15).

7. The architecture practice as a blended-workforce capability

7.1 The framework, turned on itself

Everything in this chapter has aimed outward — at the business capabilities the enterprise runs and at deciding who performs them. The same logic turns inward. Enterprise architecture and governance is itself a capability of the enterprise. It produces outcomes: decision records, target architectures, eval suites, drift reports, allocation recommendations. By AEAF's own definition, a capability is named by the outcome it delivers, independent of who or what performs it (→ see AEAF Framework §7 for the actor model). The architecture practice is therefore allocatable like any other capability — to human, agent, or hybrid performers — and the moment that is said, the framework is pointed at itself.

This is not a rhetorical flourish. It is the strongest form of internal consistency the framework can offer. A framework that governs the allocation of work to agents but exempts its own production from that discipline would be asserting a special case for itself with no principled basis. AEAF declines the exemption. The architecture function is subject to the same allocation, autonomy level (L0–L4), oversight mode, eval suite, guardrails, and retained human accountability as any capability it governs (→ see AEAF Framework §4 for the autonomy scale).

7.2 What can be allocated, and under what discipline

Several tasks inside the practice are already candidates for agent or hybrid performance. Each is an allocation in the full sense of this chapter: it carries an autonomy level on the L0–L4 scale, an oversight mode, an eval suite, guardrails, and a named accountable human. None of them transfers accountability — an agent performs the task; a human owns the outcome.

PRACTICE TASK	PLAUSIBLE AUTONOMY	OVERSIGHT MODE	EVAL / GUARDRAIL ANCHOR
Drafting Architecture Decision Records	L1-L2	In-the-loop	ADR completeness and option-coverage eval; no record closed as decided without human sign-off
Running and triaging eval suites	L2-L3	On-the-loop	Meta-evals on triage accuracy; escalation on novel failure classes
Monitoring the runtime governance loop and drift	L2-L3	On-the-loop	Drift-detection precision/recall thresholds; alert routed to the accountability owner
Proposing capability allocations and frontier moves	L1-L2	In-the-loop	Recommendation traceable to stated criteria; the human decides, the agent never commits
Drafting candidate target architecture for review	L1-L2	In-the-loop	Conformance to architecture principles; human review gate before adoption

Table 19.6 — Architecture-practice tasks as governed allocations. The pattern holds down the table: the agent does more of the drafting, watching, and triaging as it earns eval evidence; the human keeps the decision, the sign-off, and the accountability. Autonomy rises with demonstrated eval performance — not by fiat, and not all at once — exactly as it does for any business-capability allocation.

7.3 A worked illustration: the architecture-assistant agent

Consider an "architecture-assistant" agent allocated to a practice. It drafts ADRs from a decision brief and a record of the options discussed, runs the eval gate against each candidate target architecture, and watches the runtime governance loop for drift between reviews. Its allocation is graded: ADR drafting at L1 early on (the agent proposes, a human

edits and signs every record), rising to L2 once the ADR-quality eval clears a sustained threshold; eval-gate execution at L3 (the agent runs the suite and triages results on-the-loop, escalating only novel failure classes); drift monitoring at L2–L3 with on-the-loop oversight.

The Enterprise Architect is the accountability owner. The agent never adopts an architecture, never closes an ADR as decided, and never silences a drift alert on its own — those are reserved human acts. If the agent's ADR-quality eval regresses, the allocation steps back down the scale; if drift monitoring produces false-clears, oversight tightens from on-the-loop toward in-the-loop. The agent is governed by the runtime loop (→ see AEAF Framework §20) the same way any operating agent is: its outputs are evidence-backed, its behaviour is monitored, and its autonomy is earned and revocable.

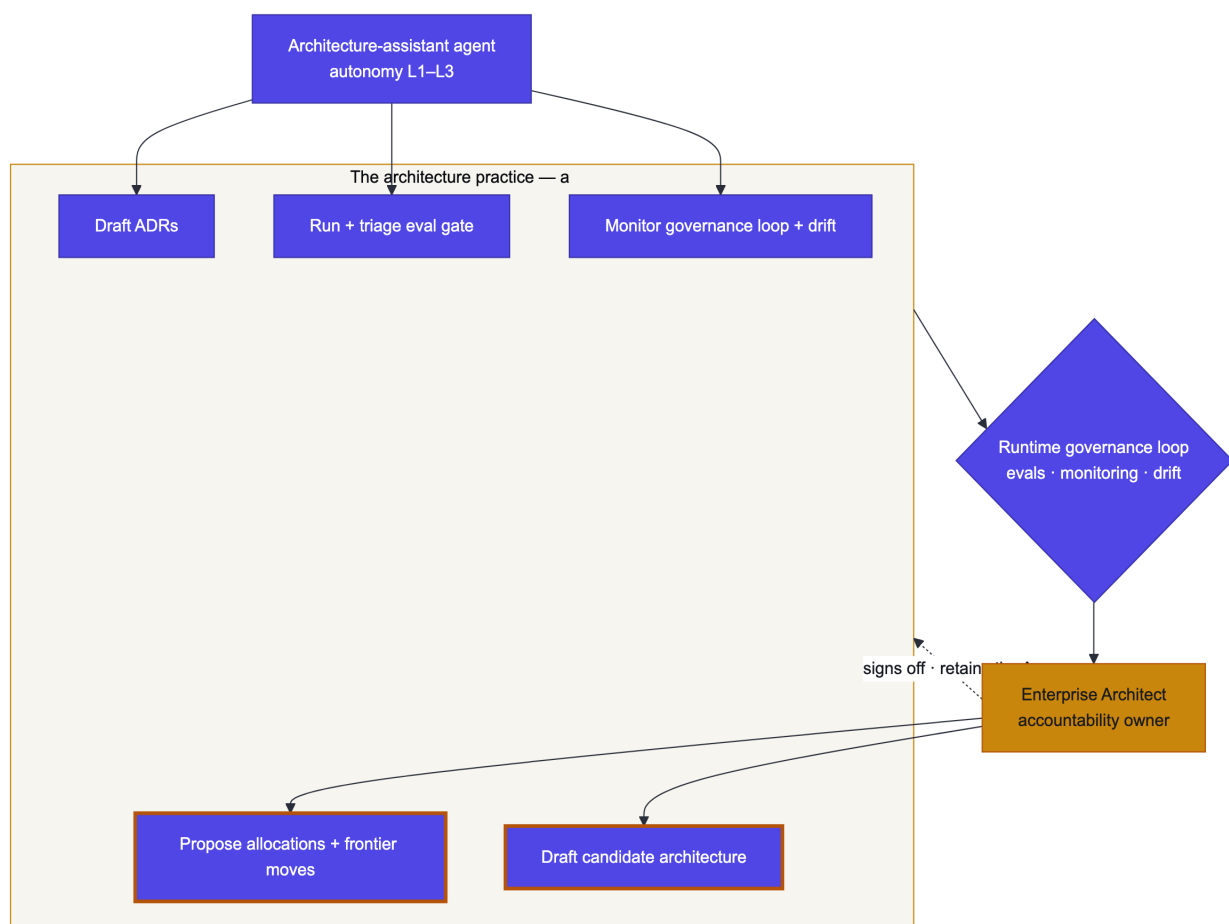


Figure 19.3 — The architecture practice as a governed capability with mixed performers. Indigo nodes are agent-performed practice tasks and the runtime governance loop that watches them; the amber node is the Enterprise Architect who holds accountability and signs off; the hybrid nodes are tasks the architect and agent perform jointly (proposing allocations, drafting candidate

architecture for review). The dotted edge is the retained A: the agent drafts, runs, and monitors, but the human signs off and remains answerable. The practice governs itself by the same loop it applies to every other capability.

7.4 Recursive-capable today, increasingly real as agents mature

Nothing in the AEAF meta-model forbids this. An Actor is human, agent, or hybrid; a Role may be agent-filled (→ see AEAF Framework §7). The practice is a capability; capabilities are allocatable (→ see AEAF in Practice §4). The framework is therefore recursive-capable as written — not as a future amendment, but today. What changes over time is not whether the recursion is permitted, but how far up the autonomy scale each practice task can credibly sit, and that is governed by evals and oversight exactly as it is everywhere else. Where agents are still weak at a task, the allocation sits low and the human carries most of it; as eval evidence accumulates, the allocation moves up under the same rules.

This is the framework's strongest form of future-proofing. AEAF does not have to predict how capable architecture-assisting agents will become. It has only to govern its own evolution by the rules it applies to everything else: allocate deliberately, grade autonomy, design oversight, maintain evals, and keep a named human accountable. A practice run this way improves its own allocations the same way it improves any capability it owns — by evidence, not by assertion.

Chapter 20 — Continuous & runtime governance

In brief. Classical architecture governance is a gate. An architecture board meets, reviews a design against principles and standards, signs an architecture contract, and the work proceeds. That model assumes the thing it approved stays the thing it approved between meetings. A probabilistic actor does not. An agent's behaviour can move while it runs — the input distribution shifts, an upstream model is updated, a knowledge corpus goes stale — and a quarterly board is blind to all of it. AEAF's answer is **dual governance**: the design-time gate is kept and extended, and a continuous **runtime governance loop** is added that governs an agent *while it operates*. This chapter sets out both halves. The design-time gate is a TOGAF Phase-G-style compliance review extended to approve, before an agent operates, its **guardrails, autonomy level, eval thresholds, oversight mode, and accountability owner**. The runtime loop enforces those guardrails at each decision point, monitors evals and drift continuously, detects and contains incidents, and adapts within bounds or escalates. An agent moves through a lifecycle state machine — **Proposed → Gated → Operating → Contained → Retired** — and a defined set of triggers (material change, widened autonomy, a failed assurance case) sends it back through the gate. The chapter defines what triggers a new design-time cycle, the incident-response sequence for autonomous action, the governance bodies and the evidence an Agentic Enterprise must keep, and how this loop binds to Intent & Assurance (Ch.17), guardrails (Ch.8), and safety (Ch.18). The setup mechanics — how to configure enforcement, drift monitors, and incident tooling — are Book 2's runtime-governance playbook; this chapter defines the governance model as a framework concept.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. Where TOGAF concentrates governance in Phase G as a gate, AEAF keeps that gate and adds a runtime loop the ADM has no equivalent for; the parallel is offered as a migration aid, not as a claim of lineage.

20.1 Why gate-only governance is blind to a system that changes between meetings

TOGAF governs through Phase G — Implementation Governance — and Phase H — Architecture Change Management. The mechanism is a gate: an architecture board or review board checks a design for compliance with principles, standards, and the target architecture; records the outcome in an architecture contract; and grants the work permission to proceed. Change is handled by raising a change request, assessing its impact, and, if it is large enough, opening a new ADM cycle. The cadence is the cadence of meetings — monthly or quarterly for the board, as-needed for review.

This works because of broken assumption A4 from the founding thesis (→ see AEAF Framework §3): *change is episodic and gated*. A deterministic system, once approved, behaves as approved until someone deliberately changes it. The gate is sufficient because nothing happens between gates. The board can sign off on a Tuesday and trust that the system on Friday is the system it saw, because the system does not rewrite its own behaviour, learn in production, or drift.

An agent breaks that assumption at machine speed. The same agent, with no redeploy and no change request, can behave differently this week than last because three things move underneath it:

WHAT MOVES UNDERNEATH AN OPERATING AGENT	EFFECT ON BEHAVIOUR	WHY A GATE CANNOT CATCH IT
The input distribution	New kinds of request arrive that the agent was never evaluated on.	The gate evaluated the agent on yesterday's cases; today's cases are different.
The model	An upstream provider updates the model the agent runs on.	The thing the gate approved is no longer the thing operating.
The knowledge corpus	The corpus the agent retrieves from goes stale, gains errors, or is poisoned.	The agent's grounding changed without the agent changing.

A gate that meets quarterly governs the snapshot it saw and nothing after it. By the next meeting, the agent has made hundreds of thousands of decisions on a distribution the gate never tested, possibly on a model the gate never approved. The gate is not wrong; it is **blind to runtime** — it has no instrument that watches the agent between sittings, and no mechanism that acts when the agent moves.

The case for dual governance follows directly. The gate is still necessary: a high-consequence agent must not reach production without a deliberate human decision that its bounds and assurance are adequate. But the gate is no longer sufficient, because the decision it makes is about a moment, and the agent lives in continuous time. AEAF therefore keeps the gate and adds a loop — a fast, continuous governance cycle that runs on the agent's clock, not the board's. The two run on different clocks, and treating them as one is the modelling error that gate-only governance makes (→ see AEAF Framework §9).

20.2 Design-time governance: the gate, extended

The design-time gate is the AEAF Method's Phase 9 — Implementation Governance (→ see AEAF Framework §9). It is the direct descendant of TOGAF Phase G and keeps everything Phase G does: a compliance review against principles and standards, an architecture contract that records what was agreed, and a sign-off that grants the work permission to proceed. AEAF does not weaken any of this. It extends it.

The extension is necessary because a classical architecture contract has nothing to say about the five things that decide whether an agent is safe to operate. Phase G asks "does this design conform to our standards?" That question is still asked. AEAF adds five more, and an agent does not cross into runtime until all five are answered and recorded.

Table 20.1 — The extended design-time gate. The first row is the classical Phase-G review; the five rows below are AEAF's additions, each of which must clear before an agent operates.

GATE CHECK	WHAT IT APPROVES	WHY IT IS AT THE GATE	OWNER OF THE ANSWER
Architecture compliance (classical Phase G)	Conformance of the design to principles, standards, and the target architecture.	The classical question, retained unchanged.	Architecture board

GATE CHECK	WHAT IT APPROVES	WHY IT IS AT THE GATE	OWNER OF THE ANSWER
Guardrails	The specific guardrails that bound this agent — input filters, output checks, tool permissions, spend limits, action vetoes — and that they enforce the intent's hard bounds and every runtime-enforced principle.	A bound the agent can cross is not a bound; the gate confirms the bounds are mechanised before the agent can act (→ see AEAF Framework §8).	Guardrail/Policy owner
Autonomy level	The autonomy level granted for this capability, on the AEAF Autonomy-Level Classification scale, and that its required controls are present.	Autonomy is granted, never assumed; the level sets which runtime controls are mandatory (→ see AEAF Framework §7).	Architecture board
Eval thresholds	The thresholds each of the five eval dimensions must clear, fixed before results are seen, and the assurance case that argues the agent clears them.	Thresholds set after results are seen are rationalisation, not assurance; the gate fixes them (→ see AEAF Framework §17).	Runtime assurance owner
Oversight mode	Whether the agent operates in-the-loop, on-the-loop, or out-of-the-loop, and the escalation path when it hits a bound or low confidence.	The oversight mode is an architectural choice tied to consequence and confidence, not an operational afterthought (→ see AEAF Framework §19).	Accountability owner

GATE CHECK	WHAT IT APPROVES	WHY IT IS AT THE GATE	OWNER OF THE ANSWER
Accountability owner	The single named human or role answerable for the agent's outcomes, recorded before the agent acts.	An agent with no named owner is an unowned action waiting to happen; accountability never transfers to an agent (→ see AEAF Framework §18).	Named at the gate

Two properties of the extended gate matter at the framework level. First, the gate produces an **agentic architecture contract**: the classical contract plus the five approvals above, signed and stored as the record of what the agent was permitted to be. This is the artefact a later incident review reads to answer "was the agent operating as approved?" Second, the gate's weight scales with consequence (→ see AEAF Framework §9). A low-consequence, reversible, in-the-loop copilot clears a light version of Table 20.1 in minutes; a high-consequence, out-of-the-loop agent that moves money clears every row with a full assurance case and a red-team result before it operates. The gate is proportionate, never skipped.

The gate's output is a handoff. When all six rows clear, the agent moves from *Gated* to *Operating* (§20.4), and governance passes from the board to the runtime loop. Nothing crosses that handoff without the contract.

20.3 Runtime governance: the loop

Once an agent operates, it is governed *while it runs*. The runtime governance loop is the AEAF Method's inner loop (→ see AEAF Framework §9): a continuous, machine-speed cycle with four activities that repeat for the agent's whole operating life. It is owned by Phase 10 — Change & Runtime Assurance — and it is the half of governance the ADM has no equivalent for.

The four activities run continuously, not in sequence on a schedule. *Enforce* fires on every agent decision; *monitor* runs against a stream of decisions; *detect/contain* fires when monitoring crosses a threshold; *adapt/escalate* fires when containment needs a response. Table 20.2 sets out each activity, what it does, and its clock.

Table 20.2 — The four activities of the runtime governance loop. Each runs continuously while the agent operates; the clocks differ by orders of magnitude.

ACTIVITY	WHAT IT DOES	WHEN IT FIRES	BINDS TO
Enforce guardrails	Checks each agent decision against its guardrails at the decision point — input filter, output check, tool permission, spend limit, action veto — and blocks any action that crosses a bound, regardless of the agent's reasoning.	On every decision (milliseconds).	Guardrails Ch.8; safety defense-in-depth Ch.18
Monitor evals & drift	Runs the online eval gate against live traffic: live metrics on all five dimensions versus the thresholds the agent cleared at the gate, plus drift detectors comparing the eval distribution to the production distribution.	Continuously, against the decision stream (seconds to minutes).	Online eval gate Ch.17; eval suite Ch.17
Detect & contain incidents	Recognises a guardrail veto, a threshold breach, a drift alarm, or an anomaly as an incident, and applies the minimum containment that stops harm — scope reduction, tool disable, or kill-switch.	On a breach, alarm, or anomaly (immediate).	Incident response §20.6; kill-switches Ch.18

ACTIVITY	WHAT IT DOES	WHEN IT FIRES	BINDS TO
Adapt within bounds or escalate	Either corrects within the agent's existing bounds (retry, fall back, re-ground) and returns it to operating, or escalates to the human owner via the designed escalation path when the situation exceeds the agent's bounds.	When containment needs a response (seconds to minutes).	Oversight & escalation Ch.19; assurance case Ch.17

Three properties of the loop deserve emphasis.

Enforcement is at the decision point, not at the gate. A guardrail that was approved at design time but checks nothing at runtime is a comment, not a control. The loop's first activity is to make every approved guardrail actually fire on every relevant decision. This is where the dual-audience principle becomes real: a principle with Enforcement = *runtime* (→ see AEAF Framework §8) was compiled into a guardrail at design time, and the runtime loop is where that guardrail does its work. Defense in depth applies — input, model, reasoning, output, and operations layers each carry controls, so a failure at one layer is recoverable by the next (→ see AEAF Framework §18).

Monitoring is the online eval gate, running live. The same five dimensions and the same thresholds the agent cleared at design time are checked continuously against production traffic (→ see AEAF Framework §17). Drift monitoring is the loop's defining job: a probabilistic actor's behaviour can move without anyone redeploying it, so the loop watches for the production distribution diverging from the eval distribution and for any live metric falling below the threshold it cleared at release. The loop also runs canary probes — eval-shaped prompts injected into live traffic at a low rate — to detect the agent behaving differently when it infers it is not being watched (→ see AEAF Framework §18).

Adaptation is bounded, escalation is designed. The loop is permitted to correct the agent *within its existing bounds* — a retry on a transient failure, a fallback to a safer path, a re-grounding against a fresh corpus. It is not permitted to widen the agent's bounds, grant more autonomy, or relax a guardrail; those are design-time decisions and require a new cycle (§20.5). When the situation exceeds the agent's bounds, the loop escalates along the path the gate approved (→ see AEAF Framework §19) — an escalation is a designed path, not a failure.

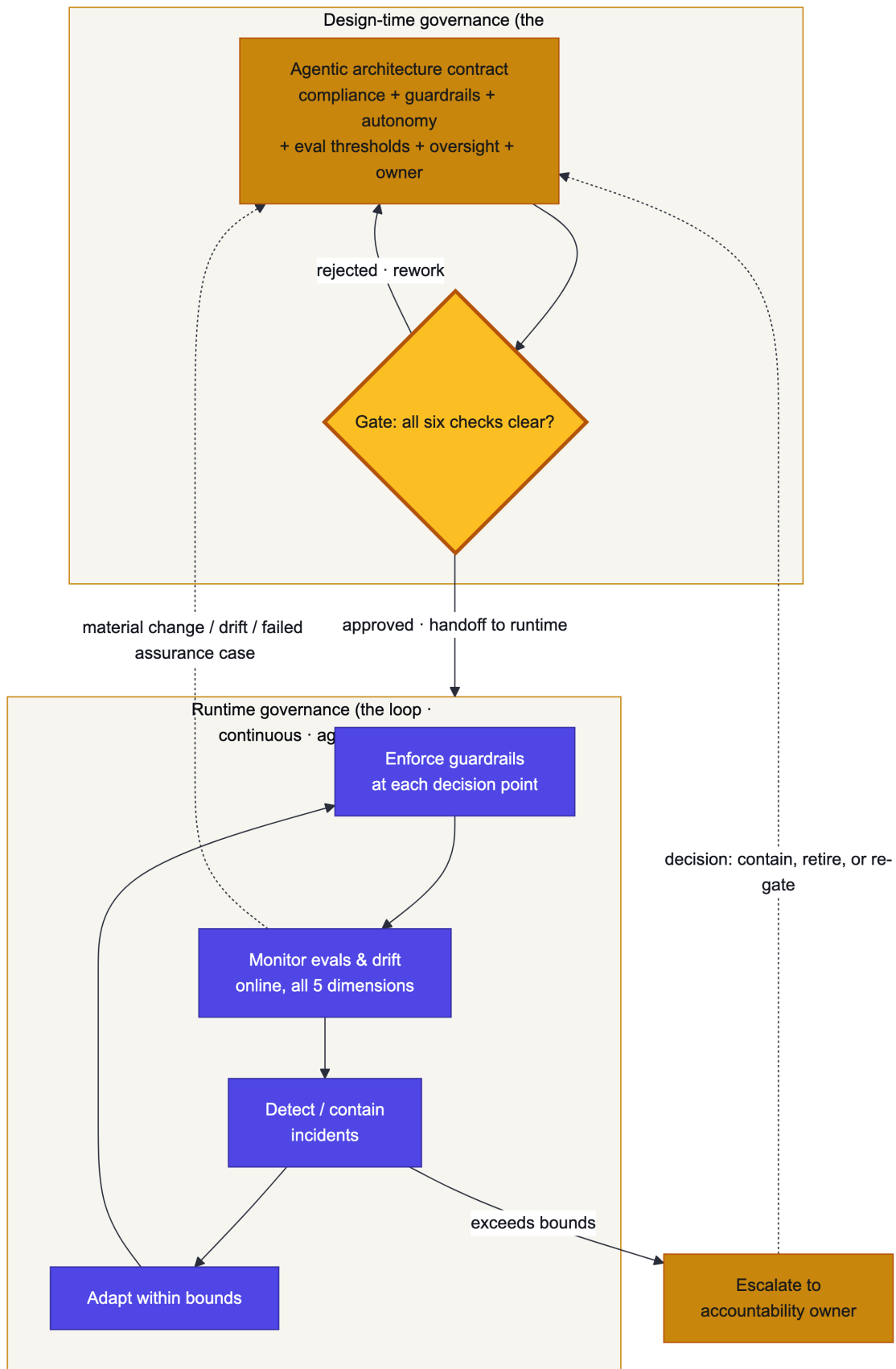


Figure 20.1 — The dual governance model. Amber is human judgment on the board's clock: the agentic architecture contract and the six-check gate that grants operation, and the accountability owner an escalation reaches. Indigo is the machine-speed runtime loop on the agent's clock: enforce, monitor, detect/contain, adapt — repeating continuously while the agent operates. The solid arrow from the gate to the loop is the handoff; the solid arrow from detect to escalate is a situation the agent cannot resolve within bounds; the dotted arrows are the return paths that re-open the gate — drift or a failed assurance case from monitoring, or an owner's decision from an escalation. The two loops meet at the contract: the gate writes it, the runtime loop reads and enforces it, and a material breach sends the agent back to it.

20.4 The agent-lifecycle state machine

Every agent under AEAF governance moves through five states. The state machine is the operational form of dual governance: it shows exactly when the gate governs (design-time states) and when the loop governs (runtime states), and the transitions are the governance events. The same five states appear at overview level in the method chapter (→ see AEAF Framework §9); here they are defined as the governed lifecycle.

Table 20.3 — The five lifecycle states and the governance that owns each.

STATE	WHAT IT MEANS	GOVERNED BY	EXIT TRANSITIONS
Proposed	The agent is designed but not yet approved; its intent, guardrails, autonomy level, evals, oversight, and owner are being prepared for the gate.	Design-time (the gate, pending)	→ Gated when the gate review begins
Gated	The agent is at the design-time gate; the six checks of Table 20.1 are being assessed and the agentic architecture contract is being signed.	Design-time (the gate)	→ Operating on approval; → Proposed on rejection (rework)

STATE	WHAT IT MEANS	GOVERNED BY	EXIT TRANSITIONS
Operating	The agent is live; the runtime loop enforces its guardrails and monitors its behaviour continuously.	Runtime (the loop)	→ Operating (the normal loop); → Contained on incident; → Retired on decommission
Contained	An incident or threshold breach has triggered containment — scope reduction, tool disable, or kill-switch; the agent is held while the response is decided.	Runtime (the loop) + human on escalation	→ Operating if corrected within bounds; → Proposed if the change is material (new cycle); → Retired if not recoverable
Retired	The agent is decommissioned — superseded, no longer needed, or judged unfit; its contract and evidence are retained for audit.	Neither (terminal)	—

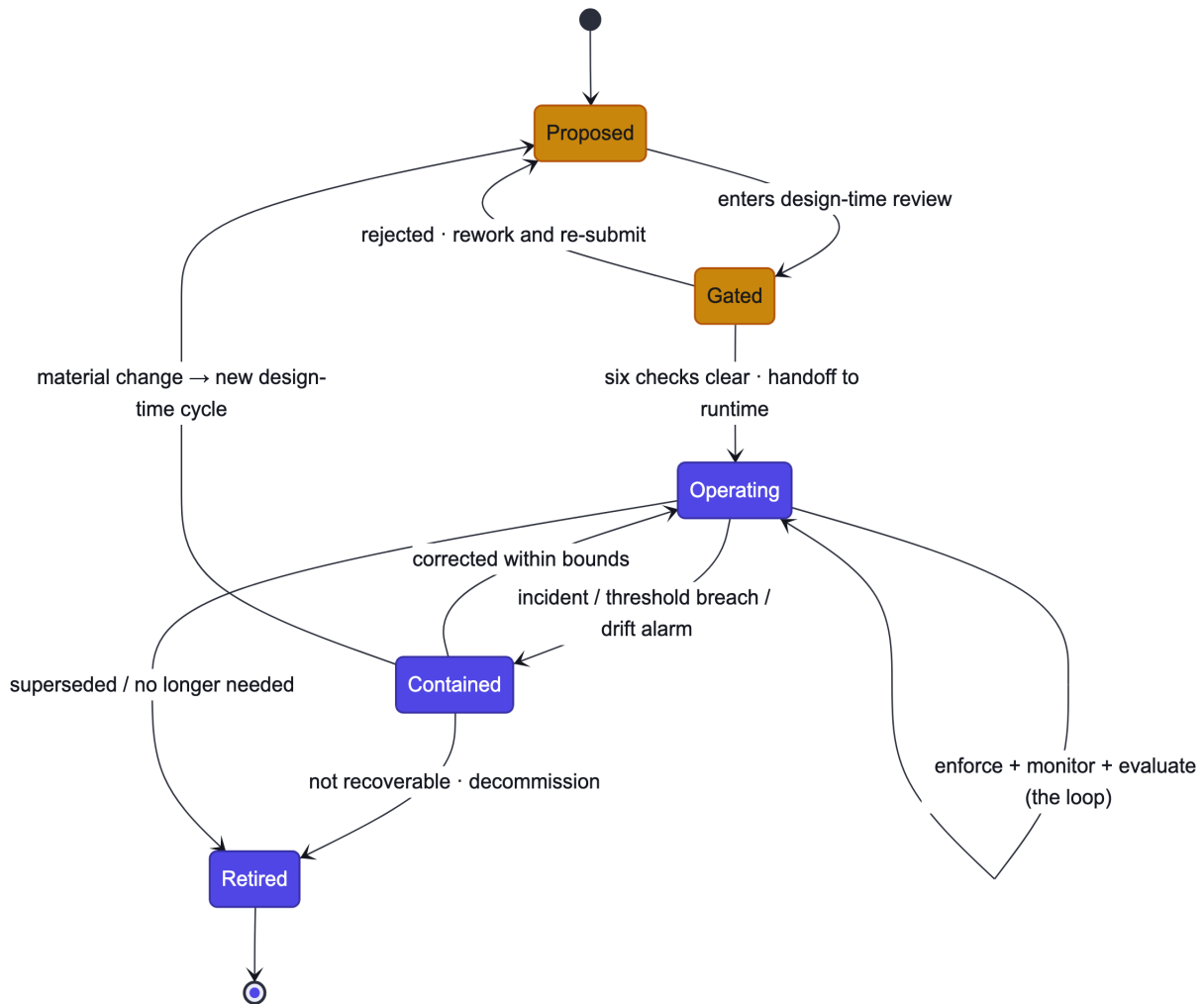


Figure 20.2 — The agent-lifecycle state machine. Amber states (Proposed, Gated) are governed by human judgment at the design-time gate; indigo states (Operating, Contained, Retired) are governed by the machine-speed runtime loop. The self-loop on Operating is the runtime loop's normal cycle of enforce-monitor-evaluate. The path Operating → Contained → Operating is the loop's routine correction within bounds; the path Contained → Proposed is the material-change return that re-opens the gate; the path Contained → Retired is the response when an agent cannot be made fit. An agent is never in Operating without a signed contract, and never enters Operating except through Gated.

The state machine makes one governance rule unmissable: **the only way into Operating is through Gated**. There is no path that lets an agent operate without first clearing the gate, and there is no path that lets the runtime loop widen an agent's bounds on its own — the loop can contain and correct, but expanding what an agent may do always routes back through Proposed and the gate. This is what keeps the runtime loop from quietly becoming an ungoverned authority.

20.5 What triggers a new design-time cycle

The runtime loop is permitted to keep an agent within its approved bounds. It is not permitted to change those bounds. When a situation requires the agent to be *more* than the contract approved, the loop cannot grant it — the matter routes back to the gate as a new design-time cycle. This is the dotted return arrow of the method cycle (→ see AEAF Framework §9), seen here as the precise set of conditions that fire it.

Three triggers re-open the gate, and they are deliberately the only three.

TRIGGER	WHAT IT IS	WHY IT CANNOT BE HANDLED IN THE LOOP
Material change	A change to what the agent is or what it runs on: a new capability added to the agent, a model swap, a substantially changed knowledge corpus, a new tool or integration.	The contract approved a specific agent on a specific model with specific tools; a material change means the operating agent is no longer the gated agent.
Widened autonomy	A request to move the agent up the autonomy scale — from suggest-only to bounded action, or from on-the-loop to out-of-the-loop.	Autonomy is granted at the gate against consequence and assurance; widening it requires re-deciding the oversight mode and the required controls.
Failed assurance case	The assurance case lapses or fails: a dimension falls below threshold and stays there, drift makes the golden set unrepresentative, or the review date passes without re-argument.	The assurance case is the argument the gate relied on to grant operation; if it no longer holds, the grant no longer holds.

The discipline here is to keep the trigger set small and the boundary sharp. A transient dip that the loop corrects within bounds is *not* a material change — it is the loop doing its job. A persistent threshold failure that breaks the assurance case *is*. The test is whether the agent can be returned to its contracted behaviour by the loop's bounded adaptation; if it can, the loop handles it; if it cannot, the gate re-opens. Confusing the two in either direction is a governance failure: treating every dip as a new cycle paralyses the agent in board meetings, and treating a broken assurance case as a dip lets an unfit agent keep operating.

When a trigger fires, the affected slice — not the whole enterprise — goes back through the design-time cycle from the point the change demands, and the agent's lifecycle state returns to Proposed (Figure 20.2). The new cycle produces an updated agentic architecture contract; only then does the agent return to Operating.

20.6 Incident response for autonomous action

An incident in the Agentic Enterprise is not only a system fault. It is an agent that acted — possibly correctly by its own reasoning — in a way that breached a bound, crossed a threshold, or caused harm. Because the actor is autonomous and fast, the response must be fast too, and it must be designed in advance, because there is no time to design it during the incident. AEAF defines a five-step sequence: **detect, contain, analyse, remediate, learn**. The first two run at machine speed inside the runtime loop; the last three run on a human investigation clock.

Table 20.4 — The incident-response sequence for autonomous action.

STEP	WHAT HAPPENS	CLOCK	WHO ACTS
Detect	The runtime loop recognises the event — a guardrail veto, a live threshold breach, a drift alarm, or an anomaly flagged by monitoring — and raises it as an incident.	Machine (immediate)	Runtime loop

STEP	WHAT HAPPENS	CLOCK	WHO ACTS
Contain	The minimum action that stops ongoing harm is applied: scope reduction (revoke a specific tool or narrow the action set), kill-switch (halt the agent's session or disable it entirely), and where the harm is spreading, global rollback to the previous model or agent version. The agent moves to Contained.	Machine (seconds)	Runtime loop; owner notified
Analyse	What happened and why: was it a guardrail gap, a drifted model, a poisoned corpus, a flawed intent, or correct behaviour the bounds failed to anticipate? The agent's decision trail (§20.7) is the primary evidence.	Human (hours)	Runtime assurance owner + accountability owner
Remediate	The fix is applied at the right level — tighten or add a guardrail, re-ground the corpus, narrow the intent, adjust the autonomy level — and routed through the gate as a new design-time cycle if the change is material (§20.5).	Human (days)	Architecture board, on the assurance owner's recommendation

STEP	WHAT HAPPENS	CLOCK	WHO ACTS
Learn	The incident updates the eval suite (the failure becomes a new golden-set case so it is caught next time), the guardrail catalog, and the principle set if a principle should have prevented it. The lesson is captured so the same incident cannot recur silently.	Human (post-incident)	Runtime assurance owner

Two properties make agentic incident response different from classical IT incident response.

Containment must be instant and graded. A human operator cannot react inside an agent's decision window — by the time a person reads the alert, the agent has acted again. Containment is therefore a machine action the loop takes autonomously, and it is graded so the response is proportionate: scope reduction for a bounded problem, a session kill-switch for a runaway, a global rollback for a spreading harm (→ see AEA Framework §18 for the kill-switch and rollback obligations). The graded kill-switch is not optional infrastructure; it is a precondition of granting autonomy in the first place, and the gate (§20.2) confirms it exists before the agent operates.

Accountability does not pause during an incident. The accountability owner named at the gate remains answerable throughout — the agent acting autonomously does not transfer answerability to the agent (→ see AEA Framework §18). The owner is notified at containment, leads the analysis, and owns the remediation decision. The "learn" step is what distinguishes governance from firefighting: an incident that contains and remediates but does not feed the eval suite and the guardrail catalog has wasted the most valuable thing it produced, which is the knowledge of a failure mode the assurance case did not cover.

20.7 Governance bodies and the evidence an Agentic Enterprise must keep

Dual governance changes who governs and what they must be able to show. Two shifts matter at the framework level: the architecture board gains a new remit, and new runtime roles join the classical EA roles.

The architecture board's new remit. The classical architecture board approves designs against principles and standards and meets on a deliberate cadence. It keeps that role. AEAF adds three things to its remit. First, it owns the **extended gate** (§20.2): granting autonomy levels, approving agentic architecture contracts, and confirming the runtime controls are in place before an agent operates. Second, it owns the **trigger decisions** (§20.5): when a runtime trigger fires, the board decides the re-gating. Third, it is accountable for the **enterprise's runtime posture** — not by sitting in the loop, which would be impossible at machine speed, but by setting the thresholds, oversight modes, and escalation policies the loop runs on, and by reviewing the runtime evidence the loop produces. The board does not watch agents; it governs the rules the agents are watched by.

New runtime roles. Three roles join the classical EA practice (defined for adoption in → see AEAF in Practice §4):

ROLE	WHAT IT OWNS	RELATIONSHIP TO CLASSICAL EA
Runtime assurance owner	The online eval gate, drift monitors, and the live assurance evidence; recommends re-gating and leads incident analysis.	New; has no classical analogue because nothing in deterministic EA needed continuous behavioural assurance.
Accountability owner	Answerability for one capability's outcomes, including the work its agents perform; named at the gate, answerable through every incident.	Extends the classical capability owner with answerability for autonomous action.
Guardrail/Policy owner	The guardrail catalog and the compilation of runtime-enforced principles into enforced guardrails.	Extends the classical principle/standards owner into machine-enforceable policy.

The evidence: decisions, not just requests. Classical IT governance keeps request logs — who asked the system to do what. The Agentic Enterprise must keep **decision logs** — what the agent decided to do, and why. The difference is the whole point: an autonomous actor's accountability rests on being able to reconstruct *its* reasoning, not only the request that prompted it. The audit trail an Agentic Enterprise must keep is set out in Table 20.5.

Table 20.5 — The runtime evidence an Agentic Enterprise must keep. The defining addition over classical governance is the decision trail: the agent's reasoning, not just the request.

EVIDENCE	WHAT IT RECORDS	WHY IT IS REQUIRED
Decision trail	For each consequential action: the input, the agent's reasoning trace, the tools it chose, the guardrail checks that fired, and the action taken.	Reconstructs why the agent acted as it did — the basis of accountability and incident analysis. Decisions, not just requests.
Guardrail-veto log	Every guardrail check, pass and fail, with the bound checked and the action taken on a veto.	Evidence in the assurance case that the agent operated inside its bounds (→ see AEA Framework §17).
Eval & drift record	Online eval results against thresholds, drift-detector output, and canary-probe results over time.	The live half of the assurance case; the signal that triggers re-gating.
Assurance-case history	Each version of the assurance case, its evidence, residual risk, and review date.	Shows the agent was fit when granted operation and remained so — or when it stopped.
Agentic architecture contract	The signed gate decision: guardrails, autonomy level, thresholds, oversight mode, owner.	The record of what the agent was permitted to be; the reference an incident review reads.
Incident record	Each incident through the five steps (§20.6): detection, containment, analysis, remediation, lesson.	Demonstrates the loop worked and the lesson was captured; satisfies regulatory and audit obligations.

Compliance metrics. Governance is only real if it is measured. Classical EA tracks an architecture-compliance rate; AEAF extends the metric set to the runtime posture, so the board can see whether the loop is doing its job without watching individual agents.

METRIC	WHAT IT TELLS THE BOARD	HEALTHY DIRECTION
Gate-pass rate	Share of agents that clear the extended gate first time.	Stable; a sudden drop signals design quality slipping.
Guardrail-veto rate	How often guardrails block an action, by agent and bound.	Non-zero (the bounds are live) but not rising (rising signals drift or a misaligned agent).
Assurance-case currency	Share of operating agents with a current, in-threshold assurance case.	→ 100%; any agent operating on a lapsed case is an ungoverned agent.
Mean time to contain	Time from incident detection to containment.	As low as machine speed allows; this is the agentic analogue of MTTR.
Re-gating rate	How often runtime triggers re-open the gate, by trigger type.	Informative, not optimised; a spike in failed-assurance triggers signals a fleet-wide problem.
Decision-trail coverage	Share of consequential actions with a complete decision trail.	→ 100%; a gap here is a hole in accountability.

20.8 How this binds to Intent & Assurance, guardrails, and safety

Continuous and runtime governance is the cross-cutting concern that the rest of the framework converges on. It does not invent the objects it governs; it operates the chain the centre defines, enforces the guardrails the principle catalog produces, and applies the controls the safety architecture specifies. Three bindings make this concrete.

To Intent & Assurance (Ch.17). The centre defines the chain *Driver* → *Intent* → *Eval* → *Assurance Case* (→ see AEAF Framework §17). Dual governance is where that chain is *read*: the design-time gate reads it forward to grant operation (the assurance case must clear the thresholds before the agent operates), and the runtime loop reads it continuously to keep or

revoke operation (the online eval gate refreshes the assurance case, and a failed case triggers re-gating). The gates defined in Ch.17 — pre-merge, pre-prod, online — are the same gates this chapter operates: the first two are the design-time gate, the third is the runtime loop.

To guardrails (Ch.8). A principle whose Enforcement attribute is *runtime* is compiled into a guardrail at design time (→ see AEAF Framework §8). The design-time gate (§20.2) confirms that compilation happened and that the guardrails enforce the intent's hard bounds; the runtime loop (§20.3) is where those guardrails actually fire, on every relevant decision. The dual-audience principle — written for a human and expressible for a machine — is only true because the runtime loop enforces the machine form. Governance is where the second audience reads the architecture.

To safety (Ch.18). Trust, safety, and accountability are the first-class concern the framework designs in, not bolts on (→ see AEAF Framework §18). Dual governance is the operating expression of that concern: defense in depth across the five layers is what the runtime loop's enforcement implements; the kill-switch, scope-reduction, and rollback obligations are what the loop's containment uses; the capability-gated control tiers are what the gate's autonomy-level check enforces; and the named accountability owner is what makes an autonomous action answerable. Safety states *what must be true*; runtime governance is *how it is kept true while the system runs*.

20.9 What this chapter establishes

Gate-only governance assumes the approved system stays approved between meetings. A probabilistic actor breaks that assumption at machine speed, so AEAF governs with two loops on two clocks. The design-time gate — TOGAF Phase G extended — approves an agent's guardrails, autonomy level, eval thresholds, oversight mode, and accountability owner in an agentic architecture contract before the agent operates. The runtime loop then governs the agent while it runs: enforcing guardrails at each decision point, monitoring evals and drift continuously, detecting and containing incidents, and adapting within bounds or escalating. An agent moves through the lifecycle Proposed → Gated → Operating → Contained → Retired, and the only way into Operating is through the gate. Three triggers — material change, widened autonomy, a failed assurance case — re-open the gate; incidents run the five-step sequence detect-contain-analyse-remediate-learn, with containment at machine speed and accountability that never pauses. The evidence an Agentic Enterprise must keep is decisions, not just requests, and the board's remit shifts from watching agents to governing the rules they are watched by. This is the cross-cutting answer to broken

assumption A4 and the binding point of the whole framework: intent, guardrails, evals, oversight, and accountability all converge here, where governance becomes a loop that runs while the system runs.

→ The how-to of standing up runtime governance — configuring enforcement points, drift monitors, incident tooling, and the runtime roles — is the runtime-governance setup playbook in AEAF in Practice §14. The construction of the eval suites the loop monitors against is the assurance playbook in AEAF in Practice §12.

Part V · Realization & reference

Chapter 21 — The AEF artifact & deliverable catalog

In brief. An architecture framework is only as usable as its artifacts — the catalogs, matrices, and diagrams in which the architecture is written down, reviewed, and kept current. This chapter is the catalog of those artifacts for the Agentic Enterprise. It does three things. First, it states the artifact taxonomy AEF inherits from classical EA — **catalogs** (lists of one kind of thing), **matrices** (relationships between two kinds of thing), and **diagrams** (a graphical view of a concern) — and confirms that AEF keeps it unchanged because it is sound. Second, it names the classical artifacts that **change** in the Agentic Enterprise, because the blended workforce puts agent actors and the things agents need into entities the artifact must now record; the Organization/Actor Catalog now lists agents, the Role Catalog gains a performer-type, and so on. Third, and chiefly, it defines **the nine new AEF artifacts** that have no classical analogue: the Agent Catalog, Model Portfolio, Knowledge-Corpus Catalog, Guardrail/Policy Catalog, Human-Agent RACI / Decision-Authority Matrix, Autonomy-Level Classification, Eval-Suite Specification, Trust & Accountability Matrix, and Capability Automation-Frontier Map. For each, this chapter gives its name, type, purpose, the meta-model entities it instantiates, the method phase that produces it, and its key contents. It then shows how the artifacts trace to one another and to the meta-model, and ends by defining the complete artifact set as the contents of the Architecture Repository for an Agentic Enterprise. The **filled templates and worked examples are not here** — they are Book 2 (→ see AEF in Practice §15). This chapter is the *definition* of each artifact; Book 2 is the *form you fill in*.

TOGAF® is a registered trademark of The Open Group. AEF is an independent framework and is neither derived from nor endorsed by The Open Group. The catalog/matrix/diagram artifact taxonomy this chapter inherits is a long-standing convention of enterprise-architecture practice that TOGAF documents; AEF adopts the taxonomy and adds nine artifacts of its own. The mapping of AEF artifacts to changed classical artifacts is offered as a migration aid for existing practices, not as a claim of lineage.

21.1 The artifact taxonomy AEAF inherits

Classical enterprise-architecture practice records an architecture in three kinds of artifact. The distinction is by *shape of information*, not by domain, and it is worth stating precisely because every AEAF artifact is one of the three.

ARTIFACT TYPE	WHAT IT IS	SHAPE	CLASSICAL EXAMPLE
Catalog	An enumerated list of all instances of one meta-model entity, each with its attributes.	One entity per row.	Application Portfolio Catalog; Principles Catalog.
Matrix	A mapping of the relationship between two meta-model entities, cell by cell.	Entity-A rows × entity-B columns.	Actor/Role Matrix; Application/Function Matrix.
Diagram	A graphical view of a concern, drawn for a stakeholder, over one or more entities.	A picture, not a list.	Business Footprint Diagram; the ADM cycle itself.
Map (AEAF usage)	A diagram or matrix whose primary subject is position and movement over a structure across time.	A structure plus an overlay of state/trajectory.	(no classical equivalent — see §21.4, artifacts 6 and 9.)

AEAF inherits this taxonomy without change. The reason is the same reason it keeps the rest of the classical machinery it can keep: the three shapes are exhaustive for how architecture information is consumed — you either list things, relate two sets of things, or draw a view of them — and a working architect already knows the shapes. AEAF adds one usage convention, the **map**, for two artifacts whose real subject is a *frontier* or a *level that moves over time* (the Automation-Frontier Map and the Autonomy-Level Classification). A map is rendered as a diagram or a matrix; it is called out separately only because its content is a trajectory, not a static set.

Every AEAF artifact, new or changed, declares its type from this list. That is what lets the repository be navigated by shape as well as by domain, and what lets a reviewer know what a given artifact is *for* before reading a line of it.

21.2 What changes in the classical artifacts

The blended workforce does not retire the classical EA artifacts. It changes what some of them must record, because the entities they list or relate now include agent actors and the things agents depend on (→ see AEAF Framework §5 for the meta-model entities named below). The change is almost always **additive**: a column, a row dimension, or an entity that the artifact already had a slot for but never had to fill with anything but a human or a deterministic component.

Table 21.1 names the classical artifacts that change and states the change. Artifacts not listed here are used as in classical practice.

Table 21.1 — Classical EA artifacts that change in the Agentic Enterprise.

CLASSICAL ARTIFACT	TYPE	WHAT CHANGES	META-MODEL DRIVER
Organization / Actor Catalog	Catalog	Now lists agent actors and hybrid teams alongside human actors. Gains an actor- type attribute (human / agent / hybrid). An agent's row links out to its Agent Catalog entry.	Actor de-anthropomorphized into Human / Agent / Hybrid Team.
Role Catalog	Catalog	Gains a permitted-performer-type attribute: which actor kinds may fill the role (human-only, agent-eligible, hybrid).	Role may now be filled by a human, an agent, or a hybrid team.
Actor/Role Matrix	Matrix	Cells may now record an agent holding a role, with a link to the autonomy level at which it holds it.	CapabilityAllocation with non-human performer.

CLASSICAL ARTIFACT	TYPE	WHAT CHANGES	META-MODEL DRIVER
Business Capability Catalog / Map	Catalog / Diagram	Each capability gains its current performer mix and autonomy level ; this is the hook the Automation-Frontier Map (artifact 9) overlays.	<code>Capability</code> ↔ <code>CapabilityAllocation</code> .
Process / Function Catalog	Catalog	A process step gains the actor-type that performs it and, where an agent, its oversight mode and escalation path.	Allocation + <code>OversightControl</code> .
Application Portfolio Catalog	Catalog	An application that hosts an agent runtime gains a <code>hostsAgents</code> link to the Agent Catalog; the application is the runtime, the agent is modelled separately.	<code>ApplicationComponent</code> hosts <code>AgentActor</code> .
Data Entity Catalog	Catalog	A data entity that is curated into a knowledge corpus gains a <code>feedsCorpus</code> link to the Knowledge-Corpus Catalog.	<code>Data Entity</code> → <code>Knowledge Corpus</code> .
Technology Portfolio Catalog	Catalog	Now includes inference platforms, model gateways, vector stores, and agent/orchestration runtimes as first-class technology components.	<code>TechnologyComponent</code> extended to the AI platform.

CLASSICAL ARTIFACT	TYPE	WHAT CHANGES	META-MODEL DRIVER
Principles Catalog	Catalog	Every principle gains the Enforcement attribute (advisory / design-time / runtime); runtime principles link to the guardrails that enforce them.	Principle made dual-audience.
Requirements Catalog	Catalog	Supplemented (not replaced) by Intent records for non-deterministic work; deterministic requirements stay as-is.	Requirement → Intent for agent work.

The pattern across the table is one move applied many times: **wherever a classical artifact assumed the performer was human or the component was deterministic, it now carries an attribute that says which, and links to the new artifact that holds the agent-specific detail.** The changed artifacts stay lightweight; the depth lives in the nine new artifacts, to which they link.

21.3 The nine new AEF artifacts at a glance

Nine artifacts have no classical analogue because the entities they record did not exist as architecture objects before the Agentic Enterprise. Table 21.2 is the master list; §21.4 defines each one. The colour column anticipates the diagram convention: **indigo** artifacts describe machine intelligence (agents, models, knowledge, guardrails, evals); **amber** artifacts carry human governance (accountability, decision authority, oversight). Two artifacts are inherently both.

Table 21.2 — The nine new AEF artifacts.

#	ARTIFACT	TYPE	PRIMARY ENTITIES INSTANTIATED	PRODUCED IN PHASE	PRIMARY DOMAIN	SIDE
1	Agent Catalog	Catalog	Agent Actor (+ links to Model, Knowledge Corpus, Guardrail, Autonomy Level, Accountability Owner)	6 · Intelligence & Agency	+I	Indigo
2	Model Portfolio	Catalog (TIME-style lifecycle)	Model	5 · Technology / 6 · Intelligence & Agency	+I	Indigo
3	Knowledge-Corpus Catalog	Catalog	Knowledge Corpus (+ link to Data Entity)	3 · Data & Knowledge	Data / +I	Indigo
4	Guardrail/Policy Catalog	Catalog + matrix	Guardrail, Policy (+ link to Principle)	6 · Intelligence & Agency (from P/8-principles)	+I / Technology	Indigo
5	Human-Agent RACI / Decision-Authority Matrix	Matrix	Capability Allocation × Actor (+ Accountability Owner, decision rights)	2 · Business & Capability	Business / +I	Amber

#	ARTIFACT	TYPE	PRIMARY ENTITIES INSTANTIATED	PRODUCED IN PHASE	PRIMARY DOMAIN	SIDE
6	Autonomy-Level Classification	Map (scale + assignment)	Autonomy Level, Capability Allocation	P · Preliminary (scale) / 2 & 6 (assignment)	+I	Indigo
7	Eval-Suite Specification	Spec / catalog	Eval Suite (+ link to Intent, Model, Agent)	6 · Intelligence & Agency	+I	Indigo
8	Trust & Accountability Matrix	Matrix	Capability × Accountability Owner × Oversight Control × Assurance	2 · Business & Capability / 9 · Governance	Business / +I	Amber
9	Capability Automation-Frontier Map	Map (diagram/matrix)	Automation Frontier (view over Capability Allocations)	2 · Business & Capability	Business / +I	Both

The ordering follows the meta-model's centre of gravity, not the method's sequence: artifacts 1–4 describe *what the machine intelligence is* (agents, the models under them, the knowledge they draw on, the guardrails that bound them); artifacts 5, 8 carry *who is accountable and who decides*; artifacts 6, 7 carry *how much independence is granted and how it is assured*; artifact 9 is the *derived view* over all allocation decisions. §21.5 shows the trace between them.

21.4 The nine artifacts defined

Each artifact below is given as a catalog definition: **name · type · purpose · meta-model entities instantiated · producing phase · key contents**. Key contents lists the columns or fields the artifact must carry — *not* the filled template, which with a worked example is Book 2 (→ see AEAF in Practice §15).

21.4.1 Artifact 1 — Agent Catalog

- **Type.** Catalog (one agent actor per row).
- **Purpose.** The authoritative register of every agent actor in the enterprise — what it is for, what powers it, what bounds it, and who answers for it. It is the agentic counterpart to the Application Portfolio Catalog, and the artifact an architect, an auditor, or an incident responder turns to first.
- **Meta-model entities instantiated.** `Agent Actor` (primary), with foreign-key links to `Model`, `Knowledge Corpus`, `Guardrail`, `Autonomy Level`, `Action/Tool`, `Eval Suite`, and `Accountability Owner`.
- **Produced in.** Phase 6 · Intelligence & Agency (→ see AEAF Framework §15). Seeded by the capability allocations of Phase 2.
- **Key contents.** Agent ID and name · purpose / objective · capability(ies) it performs · model(s) it runs (→ artifact 2) · knowledge corpora it draws on (→ artifact 3) · tools/ actions it may invoke, with permission and reversibility · guardrails that bound it (→ artifact 4) · assigned autonomy level (→ artifact 6) · oversight mode · eval suite that assures it (→ artifact 7) · accountability owner (→ artifact 8) · lifecycle status (proposed / gated / operating / contained / retired).

21.4.2 Artifact 2 — Model Portfolio

- **Type.** Catalog with a TIME-style lifecycle classification (Tolerate / Invest / Migrate / Eliminate), the model-layer counterpart to application portfolio rationalization.
- **Purpose.** The register of the underlying models in use, their cost and risk, and the rationalization stance on each — because one model powers many agents, the model is governed as a portfolio asset in its own right, not buried inside each agent. It answers "which models do we depend on, what do they cost, which are we moving off, and what runs on each."
- **Meta-model entities instantiated.** `Model` (primary), with a back-link to the `Agent Actor` s each model powers and a link to the `Eval Suite` that evaluates it.
- **Produced in.** Phase 5 · Technology (the platform and gateways) and Phase 6 · Intelligence & Agency (the rationalization stance) (→ see AEAF Framework §14, §15).

- **Key contents.** Model ID · provider · version · modality · context window · cost profile (per-call / per-token / hosting) · risk profile (data residency, provenance, dependency) · agents powered · eval results summary · **lifecycle stance** (Tolerate / Invest / Migrate / Eliminate) with rationale and review date.

21.4.3 Artifact 3 — Knowledge-Corpus Catalog

- **Type.** Catalog (one corpus per row).
- **Purpose.** The register of every governed body of content an agent retrieves from to ground its reasoning — its sources, freshness, classification, and access controls. It makes the cognitive substrate an architecture object so that what an agent *knows*, and how current and how trustworthy that knowledge is, is governed rather than assumed.
- **Meta-model entities instantiated.** `Knowledge Corpus` (primary), with a link back to the `Data Entity` (ies) it is curated from and forward to the `Agent Actor` s that retrieve from it.
- **Produced in.** Phase 3 · Data & Knowledge (→ see AEAf Framework §12).
- **Key contents.** Corpus ID and name · sources and the data entities they derive from · freshness / refresh cadence · classification (sensitivity, regulatory) · embedding model and index type · access policy (who and which agents may retrieve) · agents that draw on it · owner.

21.4.4 Artifact 4 — Guardrail/Policy Catalog

- **Type.** Catalog of guardrails, with an embedded matrix mapping policy → guardrail → enforcement point.
- **Purpose.** The register of the machine-enforceable constraints on agent behaviour and the policies they enforce. It is the artifact where a principle with `enforcement = runtime` (→ see AEAf Framework §8) is shown **compiled into** the concrete guardrails that bound an agent at runtime, with the enforcement point named. It answers "what is the agent actually prevented from doing, and where is that prevention enforced."
- **Meta-model entities instantiated.** `Guardrail` and `Policy` (primary), with a link back to the `Principle` each realizes and forward to the `Agent Actor` s each bounds.
- **Produced in.** Phase 6 · Intelligence & Agency, fed by the principle set established in the Preliminary phase and Chapter 8 (→ see AEAf Framework §8, §15).
- **Key contents.** Guardrail ID · the policy/principle it enforces · scope (which agents/capabilities) · trigger condition · enforcement point (input filter / output check / tool permission / spend limit / action veto) · fail action (block / escalate / log) · owner · the eval that tests it holds.

21.4.5 Artifact 5 — Human-Agent RACI / Decision-Authority Matrix

- **Type.** Matrix (capabilities/decisions × actors).
- **Purpose.** The split of Responsibility and Accountability between human and agent actors per capability, and — its decision-authority face — *which decisions an agent may make alone, which require a human, and which a human makes with agent input*. It is the artifact that makes the rule **accountability never transfers to an agent** operational and reviewable: an agent may be Responsible for execution; it is never Accountable.
- **Meta-model entities instantiated.** `Capability Allocation` × `Actor`, anchored by `Accountability Owner`; decision rights expressed against `Oversight Control`.
- **Produced in.** Phase 2 · Business & Capability (→ see AEAf Framework §11).
- **Key contents.** Capability / decision rows × actor columns · RACI letter per cell (R/A/C/I) with the constraint that **A is always a named human** · for each agent-Responsible cell, the autonomy level and oversight mode · decision-authority band (agent-alone / agent-proposes-human-approves / human-alone) · escalation target.

21.4.6 Artifact 6 — Autonomy-Level Classification

- **Type.** Map (a defined scale plus the per-capability assignment against it).
- **Purpose.** Two things in one artifact: the **scale** of autonomy levels the enterprise uses (e.g. L0 suggest-only → L4 autonomous-within-bounds), defined once with its required controls per level; and the **assignment** of a level to each agent capability. It stops autonomy being an ad-hoc, per-agent judgement and makes "how much independence does this agent have, and what controls did that level require" a single lookup.
- **Meta-model entities instantiated.** `Autonomy Level` (the scale) and `Capability Allocation` (the assignment); each level sets an `Oversight Control` mode.
- **Produced in.** The **scale** in the Preliminary phase (set up front, with the accountability model); the **assignment** in Phases 2 and 6 (→ see AEAf Framework §15, §19).
- **Key contents.** *Scale side:* level ID · name · definition · independent actions permitted · required controls and oversight mode at that level · gating evidence to reach it.
Assignment side: capability/agent · assigned level · effective-from date · the assurance that justifies the level.

21.4.7 Artifact 7 — Eval-Suite Specification

- **Type.** Specification (a structured spec; cases held as a catalog).
- **Purpose.** The definition of the tests, probes, and thresholds that bound and monitor an agent's or model's behaviour — the replacement for a one-time test plan with a continuously-run suite. It is the artifact that says "this is how we know the agent is

behaving within its intent, and this is the bar it must clear." It is the evidentiary backbone of the assurance case.

- **Meta-model entities instantiated.** `Eval Suite` (primary), measuring an `Intent` and assuring a `Model` / `Agent Actor` ; results feed the `Assurance Case` .
- **Produced in.** Phase 6 · Intelligence & Agency, against the Intent set at the centre (→ see AEAF Framework §17, §15).
- **Key contents.** Suite ID · the agent/model and intent it evaluates · evaluation dimensions (quality / safety / alignment / cost) · case sets and their provenance (incl. logged real cases) · expected outcomes · **pass thresholds per dimension** · run cadence · the gate it governs (design-time and/or runtime).

21.4.8 Artifact 8 — Trust & Accountability Matrix

- **Type.** Matrix (capabilities × accountability/oversight/assurance).
- **Purpose.** The single view that ties each capability to its named accountability owner, the oversight mode under which its agent operates, and the assurance evidence that currently stands behind it. Where the Human-Agent RACI (artifact 5) sets the split at design time, this matrix is the **standing answer** to "who is answerable for this agent right now, how is it overseen, and what is the live evidence it is fit to operate." It is the artifact the governance gate and an auditor read.
- **Meta-model entities instantiated.** `Capability` × `Accountability Owner` × `Oversight Control` × `Assurance` (the live evidence from `Eval Suite` + guardrail logs).
- **Produced in.** Phase 2 (initial) and Phase 9 · Implementation Governance, refreshed continuously by the runtime loop (→ see AEAF Framework §18, §20).
- **Key contents.** Capability / agent · accountability owner (named human/role) · oversight mode (in / on / out-of-the-loop) · current autonomy level · assurance status (current eval pass-rate, last incident, assurance-case review date) · residual risk · escalation path.

21.4.9 Artifact 9 — Capability Automation-Frontier Map

- **Type.** Map (a diagram or matrix over the capability model, with a time overlay).
- **Purpose.** The derived, time-versioned view of where the boundary between human-performed and agent-performed work sits across the whole capability map, and which way it is moving. It is not a primary record — it is a **view computed over the capability allocations** — and it is the artifact strategy and the board read to see the state and trajectory of the blended workforce at a glance.
- **Meta-model entities instantiated.** `Automation Frontier` — a view over the set of `Capability Allocation` s, each carrying a performer mix and autonomy level.

- **Produced in.** Phase 2 · Business & Capability, re-derived whenever allocations change; frontier *moves* are planned in Phase 8 (→ see AEAF Framework §11, §16).
- **Key contents.** The capability model as the base structure · per capability: current performer mix (human / agent / hybrid) and autonomy level · the **as-of date** of this view · the projected next frontier position and the assurance precondition for the move · trend direction per capability.

21.5 How the artifacts trace to each other and to the meta-model

The artifacts are not a flat list; they form a traceable web rooted in the meta-model entities and produced across the method phases. Figure 21.1 is the trace: each method phase (top) produces the artifacts (middle) that instantiate the meta-model entities (bottom). Reading it top-down tells a reviewer *which phase owns which artifact*; reading it bottom-up tells an auditor *which artifact records a given entity*. Indigo artifacts describe machine intelligence; amber artifacts carry human governance; the two-colour artifacts carry both.

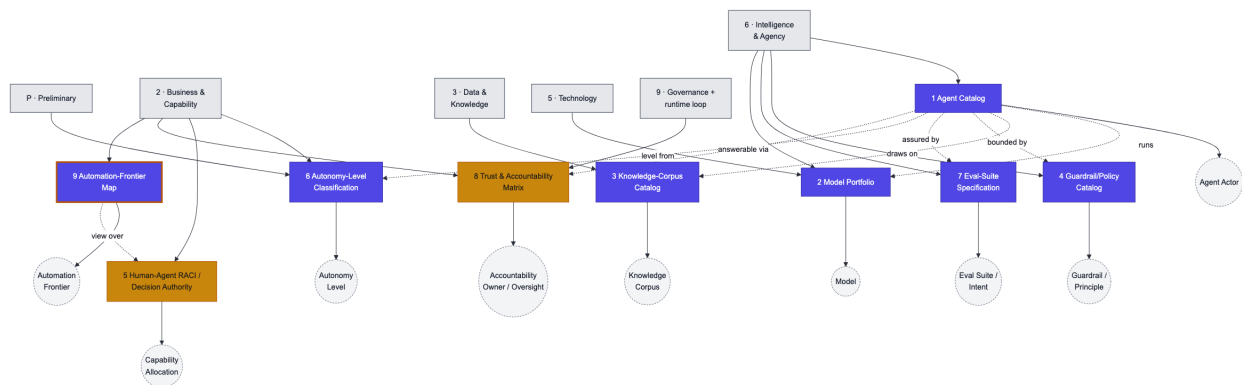


Figure 21.1 — Phase → artifact → meta-model entity trace. Each method phase (grey) produces the artifacts (indigo = machine intelligence, amber = human governance, split = both) that instantiate the meta-model entities (dashed). The dotted edges from the Agent Catalog show why it is the hub: an agent's row references its model, knowledge, guardrails, evals, autonomy level, and accountability owner — one entry per other artifact. The Automation-Frontier Map is a view computed over the allocation records the RACI holds, which is why it is derived, not primary.

Three trace properties matter for using the set:

- **The Agent Catalog is the hub.** Six other artifacts hang off an agent's row (model, knowledge, guardrails, autonomy, evals, accountability). To answer almost any question about an operating agent, you start there and follow the links. This mirrors the meta-

model, where **Agent Actor** is the entity most other agentic entities relate to (→ see AEAF Framework §5).

- **Governance artifacts are derived or refreshed, not authored once.** The Trust & Accountability Matrix (artifact 8) is refreshed by the runtime loop, and the Automation-Frontier Map (artifact 9) is re-derived whenever allocations change. They are standing views, not point-in-time documents — which is the artifact-level expression of the dual governance loop (→ see AEAF Framework §20).
- **The traceability chains of the meta-model are visible across artifacts.** The chain *Principle* → *Guardrail* → *enforcement point* lives in artifact 4; *Capability* → *Allocation* → *Actor* + *Oversight* + *Autonomy* → *Accountability Owner* spans artifacts 5, 6, 8; *Intent* → *Eval Suite* → *Assurance* lives in artifact 7 feeding artifact 8. The artifact set is how the meta-model's promised chains (→ see AEAF Framework §5) become checkable in practice.

21.6 The artifact set as the Architecture Repository for an Agentic Enterprise

The Architecture Repository is the governed store of all architecture artifacts, kept current and navigable. For an Agentic Enterprise its contents are the classical EA artifacts — with the changes in §21.2 — **plus** the nine new artifacts of §21.4. Table 21.3 states the complete set as a repository content list, organized by the classical repository partitions, so an existing TOGAF-style repository can be extended rather than rebuilt.

Table 21.3 — The Agentic Enterprise Architecture Repository: content set.

REPOSITORY PARTITION	CLASSICAL CONTENT (USED / CHANGED)	NEW AEAF CONTENT
Architecture Metamodel	The content metamodel.	Extended with the agentic entities (→ see AEAF Framework §5): Agent, Model, Knowledge Corpus, Guardrail, Allocation, Autonomy Level, Oversight, Intent, Eval Suite, Accountability Owner.
Architecture Landscape (the as-is/to-be artifacts)	Business, Data, Application, Technology catalogs/matrices/diagrams — changed per Table 21.1.	Agent Catalog (1) • Model Portfolio (2) • Knowledge-Corpus Catalog (3) • Capability Automation-Frontier Map (9).

REPOSITORY PARTITION	CLASSICAL CONTENT (USED / CHANGED)	NEW AEAF CONTENT
Standards Information Base	Standards and the Principles Catalog — principles now carry Enforcement.	Guardrail/Policy Catalog (4) · the Autonomy-Level Classification scale (6) as a governance standard.
Reference Library	Reference architectures, patterns, building blocks.	Agentic reference building blocks; eval-suite patterns; guardrail patterns.
Governance Log	Compliance assessments, architecture contracts, decisions.	Human-Agent RACI (5) · Trust & Accountability Matrix (8) · Eval-Suite Specification results (7) · agent lifecycle states and runtime-loop evidence.
Architecture Requirements	Requirements Catalog — supplemented by Intent records.	Intent register; assurance cases.

Two repository properties are specific to the Agentic Enterprise and follow from the chapters before this one:

- **Parts of the repository are now read at runtime, not only by people.** Guardrails (artifact 4), autonomy levels (artifact 6), and intent records are consumed by the running system, not merely by an architect at a review. The repository is therefore an operational dependency, not only a knowledge store — its currency and its access controls matter at machine speed (→ see AEAF Framework §20).
- **Several artifacts are living views, not versioned documents.** The Trust & Accountability Matrix and the Automation-Frontier Map reflect the current operating state and change as allocations and assurance change. A repository for the Agentic Enterprise must hold standing, derived views alongside its versioned, point-in-time artifacts — and know which is which.

The filled form of every one of these artifacts, with a worked example for a fictional enterprise, is the subject of Book 2 (→ see AEAF in Practice §15). This chapter has defined *what each artifact is and where it comes from*; that chapter shows *what one looks like once filled in*.

Note — supporting governance views are not core artifacts. AEAF defines exactly nine new core artifacts; that count is canonical and does not change. Alongside them the framework maintains lighter supporting governance views that aid decisions without being part of the core set. The chief example is the Technology Radar (→ see AEAF Framework §14): the Adopt / Trial / Assess / Hold decision log for emerging and connecting technologies, held in the Standards Information Base next to the Technology Portfolio. The Technology Radar is not a tenth core artifact — it records placement decisions, not architecture entities, and instantiates no new meta-model entity. Treat any proposal to "add a tenth core artifact" or to raise the canonical count above nine as a defect: supporting views are added freely; the nine core artifacts are fixed.

Chapter 22 — The AEAf maturity model

In brief

A maturity model tells an organization three things: where it is, where it should aim, and what the next defensible step is. AEAf's maturity model differs from a classical enterprise-architecture (EA) maturity model in one decisive way — it measures how well an organization architects, governs, and assures a **blended workforce** of human actors and agent actors, not how well it documents an IT estate. This chapter defines a five-level scale (L0 Shadow → L4 Continuously assured), assesses it across **eight dimensions** rather than collapsing it into one number, and reads the result as a **heat map** that shows imbalance, not just altitude. The model is deliberately built so that a high score on capability with a low score on assurance and accountability reads as a *risk*, not a triumph. The assessment workshop itself — facilitation, scoring rubric, evidence collection — is deferred to → see AEAf in Practice §5.

TOGAF® is a registered trademark of The Open Group. AEAf is an independent framework and is neither derived from nor endorsed by The Open Group. Where TOGAF's capability-maturity thinking measures the maturity of an architecture practice, AEAf measures the maturity of an organization's blended-workforce architecture — a different object with new dimensions.

1. Why a maturity model, and why this one is different

A maturity model serves three purposes, and AEAf's model serves all three:

PURPOSE	WHAT IT ANSWERS	HOW AEF USES IT
Self-assessment	"Where are we, honestly, today?"	An eight-dimension current-state profile, evidenced per cell, read as a heat map.
Target-setting	"Where should we be, given our consequence envelope?"	A target profile per dimension — not necessarily L4 everywhere; ambition is set against risk appetite.
Roadmap	"What is the next defensible move?"	The gap between current and target profile drives sequencing into Opportunities & Migration (→ see AEF Framework §16).

A blended-workforce maturity model is not a relabelled EA maturity model. Classical EA maturity asks how completely and how repeatably an organization describes and governs its **systems** — coverage of the landscape, repeatability of the ADM, the authority of the architecture board. Those questions assume the four conditions AEF was built to replace: human actors, human readers of architecture, deterministic systems, gated change (→ see AEF Framework §3). When agents act, the maturity question changes in four ways.

- **The unit shifts from systems to actors.** Maturity is now about how well the organization knows, allocates, bounds, and oversees its **agent actors** — not just its applications. An organization can have a mature application portfolio and a completely immature agent population.
- **Assurance replaces verification as a maturity axis.** A mature organization does not "test its agents once and ship." It runs continuous evals and holds a living assurance case. The presence and quality of that loop is a maturity dimension with no classical equivalent.
- **Governance must reach runtime.** Classical maturity peaks at a respected design-time gate. Blended-workforce maturity requires governance that runs *while the system runs* — enforcement, drift detection, incident response (→ see AEF Framework §20). Design-time maturity alone is, at the higher levels, insufficient.
- **Accountability is the safety floor, not a documentation nicety.** Because an agent can take consequential action and is never itself accountable, "who is answerable for each agent-performed capability" is load-bearing. An organization with capable agents and unclear accountability is *less* mature, and more dangerous, than one with modest agents and clear ownership.

The consequence: AEAF deliberately refuses a single maturity number. A single number lets a high capability score hide a missing assurance loop. The model is multi-dimensional precisely so that **imbalance is visible** — the most common and most dangerous failure mode being capability and ambition that run ahead of governance, assurance, and accountability.

2. The AEAF maturity levels

AEAF uses a five-level scale, L0 through L4. The levels describe the **organization's blended-workforce architecture as a whole**; §3 then resolves each level per dimension, because real organizations sit at different levels on different dimensions.

LEVEL	NAME	ONE-LINE CHARACTERIZATION
L0	Shadow	Agents are used, but unknown to the architecture. No catalog, no owner, no bounds.
L1	Piloting	Agents are deliberate but isolated. Each pilot is governed by hand, nothing is reusable.
L2	Governed	Agents are cataloged, owned, and gated before they operate. Governance is real but design-time only.
L3	Blended	Capability allocation is explicit and managed; design-time and runtime governance both operate.
L4	Continuously assured	The blended workforce is architected, the automation frontier is planned, and assurance runs continuously and feeds the cycle.

The defining transitions are: **L0→L1** the organization stops pretending agents are not there; **L1→L2** governance becomes systematic rather than per-pilot; **L2→L3** governance reaches runtime and allocation becomes a managed design act; **L3→L4** assurance becomes continuous and closes the loop back into the method.

2.1 The five levels in detail

The table below characterizes each level against the questions that matter for a blended workforce. Read a row as a portrait of an organization at that level.

ATTRIBUTE	L0 SHADOW	L1 PILOTING	L2 GOVERNED	L3 BLENDED	L4 CONTINUOUSLY ASSURED
How agents are used	Ad-hoc, by individuals, often via consumer tools; invisible to IT and architecture.	A handful of sanctioned pilots, each bespoke; mostly suggest-only or in-the-loop.	Agents in production for defined capabilities; autonomy levels assigned but conservative.	Agents and humans share capabilities by design; autonomy is graded per capability and moves with trust.	Agents operate across the frontier at the right autonomy level; re-allocation is routine and planned.
Governance posture	None. No gate, no policy, no record.	Manual, per-pilot sign-off; no reusable policy.	A real design-time gate : agents are reviewed and approved before operating; a Guardrail/Policy Catalog exists.	Design-time gate plus a runtime governance loop: guardrails enforced, policies machine-enforceable.	Dual governance is continuous and instrumented; policy-as-guardrail is the norm; gate decisions are evidence-based.

ATTRIBUTE	L0 SHADOW	L1 PILOTING	L2 GOVERNED	L3 BLENDED	L4 CONTINUOUSLY ASSURED
Assurance posture	None. Behaviour is unobserved.	Spot checks; maybe a demo. No eval suite.	Pre-deployment evals exist; little or no continuous monitoring.	Eval suites per capability and runtime monitoring; an assurance case is maintained.	Continuous assurance: evals, drift detection, and guardrail logs feed a living assurance case that gates change.
Accountability clarity	Unknown. No named owner for agent-performed work.	Implicit — “the project team.” Not recorded per capability.	A named accountability owner per agented capability, recorded.	Trust & Accountability Matrix maintained; oversight mode set per capability.	Accountability and oversight are live, audited, and re-confirmed as autonomy changes.
How allocation is decided	Not decided; it happens.	Pilot-by-pilot, opportunistically.	Per capability at the gate, but largely static once set.	Capability allocation is an explicit, revisited design decision; the automation frontier is mapped.	Frontier moves are planned, sequenced, and governed as the central transformation lever.
Typical risks	Data leakage, ungoverned action, no audit trail, “who approved this?” with no answer.	Pilots that never scale; inconsistent controls; learning not captured; over-trust in a demo.	False comfort: a strong gate but blind to runtime drift and post-deployment failure.	Complexity of the dual loop; assurance cost; allocation churn if trust signals are weak.	Diminishing returns chased; over-engineering low-consequence agents; complacency in a mature loop.

2.2 Reading the levels against the trust-gated rollout

The levels are not arbitrary stages; they track the **trust-gated rollout** the method already prescribes — pilot under tight oversight, widen autonomy as assurance accrues (→ see AEA Framework §16, Ch.16). An organization earns the right to higher autonomy by demonstrating assurance, and the maturity level reflects how systematically it does so.

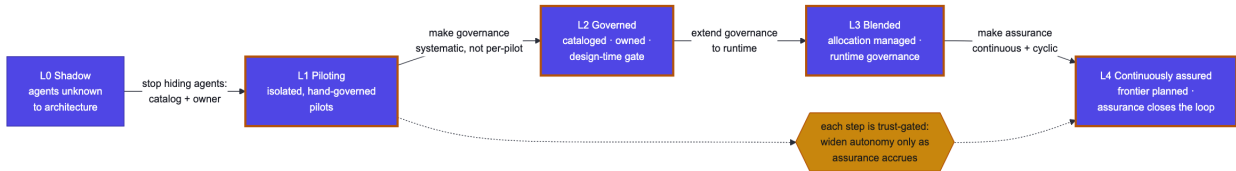


Figure 22.1 — The AEA maturity ladder. The indigo-to-hybrid progression shows agent-maturity advancing from invisible (L0) to fully assured (L4); the amber gate is the human-governance condition on every advance — autonomy is widened only as assurance is earned. The rungs are governance and assurance milestones, not adoption-volume milestones: more agents at L0 is not higher maturity.

A crucial reading of Figure 22.1: progress is **left-to-right by governance and assurance, not by agent count**. An organization with two hundred shadow agents is at L0, not ahead of one with five governed agents at L2. This is the model's central guard against mistaking activity for maturity.

3. The eight maturity dimensions

A single level hides imbalance. AEA therefore assesses across eight dimensions, chosen so that the dangerous gaps — capability outrunning assurance, ambition outrunning accountability — show up as a jagged profile. Each dimension is scored L0–L4 independently.

#	DIMENSION	WHAT IT MEASURES	ANCHORED IN
D1	Strategy & ambition	Is there a stated human-agent ambition and a consequence/risk envelope, or is adoption opportunistic?	Vision (→ §10)

#	DIMENSION	WHAT IT MEASURES	ANCHORED IN
D2	Capability allocation	How deliberately is human/agent/hybrid allocation decided and revisited; is the automation frontier mapped?	Business & capability (→ §11), Ch.7
D3	Intelligence & Agency architecture	Are models, agents, knowledge corpora, and orchestration architected as first-class entities, or assembled ad-hoc?	+I domain (→ §15)
D4	Assurance & evaluation	Do eval suites exist, run continuously, and feed a living assurance case — or is behaviour unobserved?	Intent & Assurance (→ §17)
D5	Governance (design-time + runtime)	Is there a design-time gate, a runtime enforcement loop, both, or neither?	Continuous & runtime governance (→ §20)
D6	Accountability & safety	Is there a named accountability owner and oversight mode per agented capability; is the threat surface managed?	Trust, safety & accountability (→ §18)
D7	Workforce & operating model	Are oversight, escalation, decision authority, and the human transition (reskilling, role change) designed?	Human-agent operating model (→ §19)

#	DIMENSION	WHAT IT MEASURES	ANCHORED IN
D8	Platform & technology	Is there a managed AI platform — inference, model gateway, vector stores, agent runtime, LLMOps — or scattered point tools?	Technology architecture (→ §14)

3.1 The maturity matrix (dimension × level)

The matrix below is the assessment instrument. To assess, locate the cell that the organization's **evidence** supports for each dimension (§4), not the cell it aspires to. The result is a profile across the eight rows.

DIMENSION	L0 SHADOW	L1 PILOTING	L2 GOVERNED	L3 BLENDED	L4 CONTINUOUSLY ASSURED
D1 Strategy & ambition	No stated ambition; agents appear bottom-up.	Ambition stated for pilots; no risk envelope.	Ambition and a consequence/risk appetite documented.	Ambition tied to capability roadmap; frontier targets set.	Ambition, risk envelope, and frontier plan reviewed each cycle.
D2 Capability allocation	Not decided; allocation just happens.	Per-pilot, opportunistic.	Decided at the gate per capability; mostly static.	Explicit, revisited; automation frontier mapped.	Frontier moves planned, sequenced, governed.
D3 Intelligence & Agency arch.	No +I view; models/agents invisible.	Agents built ad-hoc per pilot; no catalog.	Agent Catalog + Model Portfolio maintained.	Orchestration, knowledge corpora, guardrails architected together.	+I domain managed as a portfolio with lifecycle (TIME-equivalent).

DIMENSION	L0 SHADOW	L1 PILOTING	L2 GOVERNED	L3 BLENDED	L4 CONTINUOUSLY ASSURED
D4 Assurance & evaluation	Behaviour unobserved.	Spot checks / demos.	Pre-deployment eval suites.	Eval suites + runtime monitoring; assurance case maintained.	Continuous evals + drift detection feeding a living assurance case.
D5 Governance (design + runtime)	None.	Manual per-pilot sign-off.	Real design-time gate; policy catalog.	Design-time gate + runtime enforcement loop.	Dual governance continuous, instrumented, evidence-based.
D6 Accountability & safety	No named owner; threat surface unmanaged.	Implicit ownership ("the team").	Named accountability owner per capability.	Trust & Accountability Matrix + oversight mode per capability.	Accountability live, audited, re-confirmed as autonomy changes.
D7 Workforce & operating model	No oversight/escalation design.	Oversight by improvisation.	Oversight modes assigned; escalation paths defined.	Decision authority + human transition planned.	Operating model adapts as the frontier moves; teaming failure modes designed against.
D8 Platform & technology	Consumer tools, shadow SaaS.	Point tools per pilot.	Shared AI platform emerging (gateway, vector store).	Managed platform: agent runtime, LLMOps, observability.	Platform self-instruments assurance and governance signals.

Table 22.1 — The AEAF maturity matrix. Each dimension is scored independently against its row. A balanced target is a contiguous shape; a jagged shape signals risk (§5). This is the instrument the assessment workshop fills cell by cell with evidence — the facilitation method is in → see AEAF in Practice §5.

4. How to assess: evidence per cell

A maturity claim is only as good as its evidence. AEAF scores a dimension at level *L* only when the organization can produce the **artifact or observable practice** that level requires — not a slide that asserts it. The AEAF artifacts (→ see AEAF Framework §21, Ch.21) are the natural evidence base.

DIMENSION	EVIDENCE THAT A LEVEL IS GENUINELY REACHED
D1 Strategy & ambition	A documented human-agent ambition and a written consequence/risk appetite; for L3+, a frontier target in the roadmap.
D2 Capability allocation	Capability allocations recorded per capability; for L3+, a maintained Capability Automation-Frontier Map .
D3 +I architecture	A populated Agent Catalog and Model Portfolio ; for L3+, Knowledge-Corpus and Guardrail/Policy catalogs in use.
D4 Assurance & evaluation	Eval-Suite Specifications that actually run; for L3+, monitoring dashboards and a dated, living assurance case.
D5 Governance	Gate decision records; for L3+, runtime enforcement logs and machine-enforceable policy in production.
D6 Accountability & safety	A Trust & Accountability Matrix with one named owner per capability; threat-surface review evidence.
D7 Workforce & operating model	Defined oversight modes and Human-Agent RACI ; for L3+, escalation paths exercised and a human-transition plan.

DIMENSION	EVIDENCE THAT A LEVEL IS GENUINELY REACHED
D8 Platform & technology	Inventory of the AI platform components in managed use; for L4, automated assurance/governance telemetry.

The evidence rule (binding for assessment): if the artifact cannot be produced, the level is not reached, regardless of self-report. Score the level the evidence supports. This rule is what prevents an L0-Shadow organization from claiming L2 because someone "intends to" catalog its agents.

4.1 Reading the result as a heat map

The eight scores are read together, not averaged. Plotting them as a heat strip (one cell per dimension, shaded by level) turns the profile into a shape the eye reads instantly:

- A **contiguous, even shape** (e.g. all dimensions at L2–L3) is healthy: the organization is balanced, and the next move is a coordinated step up.
- A **jagged shape** is the signal the model exists to surface. The two diagnostic patterns are below.

PATTERN	PROFILE	WHAT IT MEANS	ACTION
Capability ahead of control	D2/D3/D8 at L3, D4/D5/D6 at L0–L1	The organization can build and run agents but cannot assure, govern, or own them. The most common and most dangerous gap.	Freeze autonomy expansion; raise assurance, governance, and accountability before widening the frontier.
Control ahead of capability	D4/D5/D6 at L2–L3, D2/D3 at L0–L1	Heavy governance over almost no agent activity. Safe but value-starved; governance is theatre.	Right-size governance to consequence; invest in the +I architecture so there is something worth governing.

The Vega-Lite specification below renders an eight-dimension heat strip with explicit dimensions so the long dimension labels do not collide.

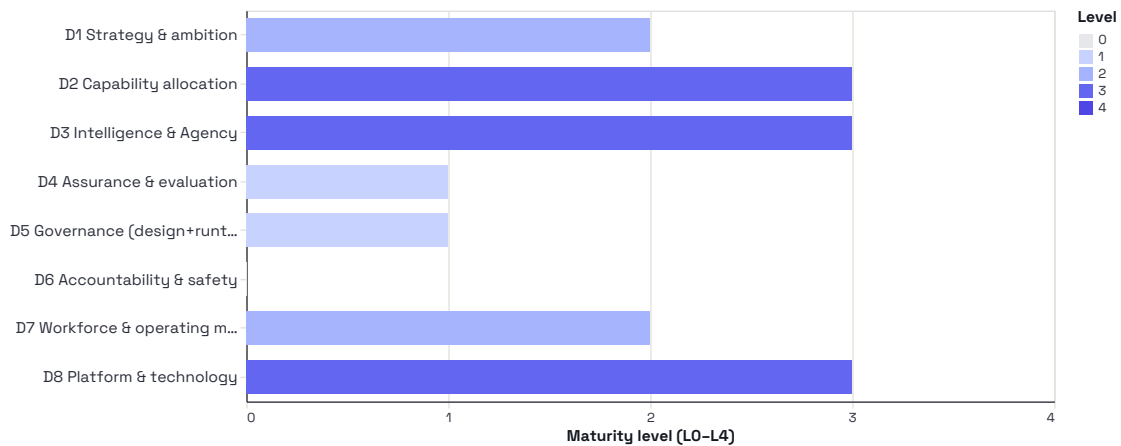


Figure 22.2 — A maturity heat strip for a "capability ahead of control" profile. D2/D3/D8 sit at L3 while D4/D5/D6 sit at L0–L1: the organization builds and runs agents faster than it assures, governs, or owns them. The jaggedness is the finding; the indigo intensity rises with level so the cold (light) cells on assurance, governance, and accountability are visible at a glance.

5. Using maturity: where to start and how to set a target

Maturity is not a trophy; it is an input to two later decisions. First, it sets **where you start** an AEAf adoption. Book 2 uses the current-state profile from this chapter to choose the entry point and the minimum viable subset of the method to adopt — a Shadow (L0) organization begins by getting agents into a catalog and assigning owners, while a Governed (L2) organization begins by extending governance into runtime (→ see AEAf in Practice §5 for the assessment workshop, §6 for the minimum viable adoption).

Second, maturity drives the **roadmap**, because the gap between current and target profile is precisely the input to Opportunities & Solutions and Migration Planning (→ see AEAf Framework §16). Three rules govern target-setting:

- **The target is not L4 everywhere.** Maturity is matched to the **consequence envelope**. A low-consequence, reversible, in-the-loop set of agents does not need L4 governance; over-engineering it is the L4 risk named in §2.1. Set each dimension's target to the level its consequence warrants.
- **Close control gaps before capability gaps.** Where the profile shows capability ahead of control (§4.1), the target *must* raise D4/D5/D6 before D2/D3/D8 advance further. Widening autonomy on top of a missing assurance loop is the failure the trust-gated rollout exists to prevent (→ see AEAf Framework §16, Ch.16).

- **Move one rung at a time, gated by assurance.** The level transitions in Figure 22.1 are sequential for a reason: each rung's governance and assurance posture is the foundation the next rung stands on. A roadmap that jumps L1→L4 is a roadmap that has not earned the trust it assumes.

Read this way, the maturity model and the trust-gated rollout are the same discipline seen from two angles. The rollout earns autonomy capability by capability as assurance accrues; the maturity model reports, across eight dimensions, how systematically the organization does so — and refuses to let a high capability score paper over a missing assurance loop.

Chapter 23 — Adopting AEAF alongside / migrating from TOGAF

In brief. An enterprise that already runs a TOGAF practice does not have to discard it to adopt AEAF. The opposite is true: roughly three-quarters of a mature TOGAF practice carries straight across — the cycle, the domains, the repository, governance as a discipline, capability-based planning, contracts, and the principle structure all survive, either unchanged or extended (→ see AEAF Framework §2). What changes is the workforce — humans now work alongside AI agents that decide and act at machine speed against probabilistic models — and that single shift propagates through specific TOGAF constructs in predictable ways. This chapter is the bridge for an existing practice. It gives the four master correspondences (method phases, domains, artifacts, governance), then a three-step incremental adoption path that lets a TOGAF shop move at its own pace: **first** add the reworked Actor model and the +I domain to work it is already doing; **second** introduce Intent & Assurance for the probabilistic work that requirements-and-testing cannot hold; **third** stand up the runtime governance loop. It says plainly which existing artifacts to extend versus create new, how to run both frameworks side by side during transition, and answers the objections a TOGAF practice will and should raise. Tailoring presets and adoption archetypes are out of scope here and belong to Book 2 (→ see AEAF in Practice §2 and §3).

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. AEAF answers and maps to TOGAF concepts to help an existing practice cross over; it is not a TOGAF product, profile, adaptation, or certified derivative. Where TOGAF uses the ADM, AEAF uses the AEAF Method. The phase-to-ADM correspondences in this chapter are migration aids for practitioners who already know the ADM, not claims of lineage.

23.1 The reassurance: this is extension, not replacement

The first thing a TOGAF practice needs to hear is the honest net assessment, because the fear is reasonable: that adopting AEAF means throwing away years of repository content, principle catalogs, capability models, and a trained team. It does not. AEAF was built by mapping TOGAF construct by construct and asking of each one whether the agentic enterprise leaves it standing. The answer for most constructs is yes.

The gap analysis classifies every mapped TOGAF element into one of four states (→ see AEAF Framework §2):

STATE	MEANING	WHAT IT IMPLIES FOR YOUR PRACTICE
Holds	Works unchanged in the agentic enterprise.	Keep your existing asset and method as-is.
Bends	Sound, but needs a defined extension.	Keep the asset; add the agentic attribute, axis, or view.
Breaks	A core assumption fails; needs replacement.	Keep the discipline; replace the mechanism.
Missing	The concern has no TOGAF analogue at all.	Create new — this is genuinely additive work.

The distribution is the reassurance. Of the elements mapped, the large majority *hold* or *bend*. The method as a cycle, the BDAT domains, the Architecture Repository, the Standards Information Base, governance as a discipline, capability-based planning, architecture contracts, building blocks, the baseline→target→gap→roadmap move, and the four-part principle structure all carry forward. The genuine *breaks* cluster at exactly three points, and they share one root cause: TOGAF assumes deterministic systems that humans operate on episodic, gated change cycles. The genuinely *missing* concerns — accountability for agent actions, the human-agent collaboration model, evaluation-instead-of-testing, and the cross-cutting Trust, Safety & Accountability concern — are net-new, but they are additions to the side of your practice, not demolitions of it.

The practical consequence: **your investment is mostly safe**. You will extend more than you replace, and you will create some new things that have no predecessor. The rest of this chapter tells you, precisely, which is which.

23.2 The master correspondence: AEF ↔ TOGAF

This section gives the four mappings a TOGAF practice needs to orient itself. They are migration aids — a way to find the AEF construct that answers the TOGAF construct you already know.

23.2.1 Method phases ↔ ADM phases (the migration-bridge table)

The AEF Method keeps the shape of the ADM and makes three changes: the centre becomes **Intent & Assurance** rather than Requirements Management; a new phase — **Intelligence & Agency** — is added for the population of models, agents, knowledge, guardrails, and evaluations; and governance gains a continuous runtime loop alongside the design-time gate (→ see AEF Framework §9). Table 23.1 is the bridge a practitioner who knows the ADM reads first.

Table 23.1 — AEF Method phase ↔ TOGAF ADM phase. Read it as "the ADM phase you run today maps to this AEF phase, which does the same work plus the agentic addition named in the last column." One phase (6) has no ADM analogue; one centre (Intent & Assurance) replaces a centre (Requirements Management).

AEF METHOD PHASE	ADM ANALOGUE	SAME WORK AS THE ADM	AGENTIC ADDITION
Preliminary	Preliminary	Stand up the capability; establish principles, governance, repository.	Establish the agentic operating context, the autonomy-level scheme, and the accountability model up front; author principles for dual audience.
1 • Vision	A	Drivers, stakeholders, scope, vision, approval.	State the human-agent ambition and the consequence/risk envelope for autonomy.
2 • Business & Capability	B	Capabilities, value streams, processes, organization.	Capability allocation (human/agent/hybrid) and the automation-frontier map as primary outputs.

AEAF METHOD PHASE	ADM ANALOGUE	SAME WORK AS THE ADM	AGENTIC ADDITION
3 • Data & Knowledge	C (data)	Conceptual/logical/physical data, governance, MDM.	Add the cognitive substrate : knowledge corpora, embeddings, memory, graphs.
4 • Application	C (app)	Application portfolio, integration, services.	Design for non-determinism : bounded outputs, fallback, handoff, observability.
5 • Technology	D	Infrastructure, platforms, security.	The AI platform : inference, model gateways, vector stores, agent runtime, LLMOps.
6 • Intelligence & Agency	— (new)	—	The whole phase. Architect models, agents, orchestration, guardrails, evals, autonomy.
7 • Opportunities & Solutions	E	Work packages, transition architectures, value increments.	Sequence by trust earned : tight oversight first, widen autonomy as assurance accrues.
8 • Migration Planning	F	Sequencing, dependencies, risk, business case.	Plan frontier moves and the human transition (reskilling, role change).
9 • Implementation Governance	G	Compliance reviews, architecture contracts, the design-time gate.	The gate also approves guardrails, autonomy levels, eval thresholds, and the handoff to the runtime loop .

AEAF METHOD PHASE	ADM ANALOGUE	SAME WORK AS THE ADM	AGENTIC ADDITION
10 • Change & Runtime Assurance	H	Monitor, assess impact, evolve, trigger new cycles.	Owns the continuous runtime loop : enforcement, drift/eval monitoring, incident response, bounded adaptation, escalation.
Intent & Assurance Management	Requirements Management (centre)	A persistent concern threading every phase.	Replaces pure requirements: declare intent and bounds, then evaluate against eval suites; deterministic requirements still apply unchanged.

23.2.2 Domains ↔ domains (BDAT → BDAT+I)

The four TOGAF domains hold and remain necessary — Business, Data, Application, and Technology are all valid and carry forward. What breaks is the *claim that the four-domain model is complete*. BDAT has no home for agency, cognition, autonomy, or the model/knowledge/eval assets that now drive behaviour; spreading those across the four domains loses them as a coherent design concern. AEAF adds a fifth domain — **I, Intelligence & Agency** — and writes the set as **BDAT+I** (→ see AEAF Framework §6).

TOGAF DOMAIN	AEAF STATE	WHAT CARRIES; WHAT IS ADDED
Business (B)	Holds, extended	Capabilities, value streams, processes carry. Add capability allocation and the automation frontier.
Data (D)	Holds, extended	Transactional data models carry. Add the cognitive substrate (corpora, embeddings, memory, graphs).

TOGAF DOMAIN	AEAF STATE	WHAT CARRIES; WHAT IS ADDED
Application (A)	Holds, extended	Application components and integration carry. Add Agent and Model as first-class component types.
Technology (T)	Holds, extended	Compute, network, storage, security carry. Add the agentic technology stratum and agent-aware security.
— (no analogue)	Intelligence & Agency (+I) (new)	Owns agents, models, capability allocation, autonomy levels, knowledge corpora, eval suites, and the human-agent collaboration model.

The +I domain is a domain, not a layer of the Application domain. Modelling agents as applications loses their decision-making, non-deterministic, runtime-governed nature, which is exactly the nature that needs its own home.

23.2.3 Artifacts ↔ artifacts (extend the classical set, add nine)

Most classical EA artifacts bend: a catalog is still the right shape, a matrix is still the right shape, but each is incomplete without the new entities. AEAF adds nine named artifacts to the classical catalogue (→ see AEAF Framework §21; templated in AEAF in Practice §15). Table 23.2 maps each new artifact to the closest classical artifact it extends or sits beside.

Table 23.2 — AEAF artifacts against their closest TOGAF artifact. "Extends" means the classical artifact gains the new entity or axis; "New beside" means the artifact has no classical predecessor and is created fresh.

AEAF ARTIFACT	CLOSEST TOGAF ARTIFACT	RELATIONSHIP
Agent Catalog	Application Portfolio Catalog	New beside — agents are a distinct actor type, not applications.
Model Portfolio	Technology Portfolio Catalog / TIME portfolio	New beside, TIME-equivalent for models.

AEAf ARTIFACT	CLOSEST TOGAF ARTIFACT	RELATIONSHIP
Knowledge-Corpus Catalog	Data Entity Catalog	Extends the data catalogs with grounding content (provenance, freshness, access).
Guardrail/Policy Catalog	Standards Information Base	New beside — machine-enforceable policies and their enforcement points.
Human-Agent RACI	Architecture RACI / responsibility matrix	Extends RACI from architecture work to agent actions in production.
Autonomy-Level Classification	—	New beside — no classical analogue.
Eval-Suite Specification	Requirements / acceptance test specification	New beside — evaluate-against-bounds replaces test-to-spec for agent work.
Trust & Accountability Matrix	—	New beside — capability → accountability owner → oversight mode → assurance.
Capability Automation-Frontier Map	Function/Application matrix	Extends the capability/performer view with the human/agent boundary and its trajectory.

Beyond these nine, classical catalogs, matrices, and diagrams *bend* in place: the function/application matrix gains a Capability-Allocation expression; communication and process diagrams gain an **Oversight/Escalation flow** view and an **autonomy-boundary** view; the principle catalog gains an **Enforcement** attribute on every principle (advisory / design-time / runtime guardrail).

23.2.4 Governance ↔ governance (+ the runtime loop)

This is the deepest change and the one a TOGAF governance lead must understand clearly. TOGAF governance assumes change is episodic and reviewed *before* go-live; the architecture board, compliance reviews, and architecture contracts are all built around a design-time gate. That gate holds and stays. What breaks is the assumption that the gate is *enough*:

agents act and adapt continuously after deployment, faster than any review board can convene, and design-time compliance review cannot see behaviour that only exists at runtime (→ see AEAF Framework §20).

AEAF's answer is **dual governance** — keep the design-time gate, and pair it with a runtime loop.

TOGAF GOVERNANCE CONSTRUCT	AEAF STATE	AEAF RESPONSE
Implementation Governance (Phase G)	Breaks	Keep the design-time gate; add runtime policy enforcement and continuous assurance.
Architecture Change Management (Phase H)	Breaks	Replace episodic change handling with continuous assurance : behaviour re-evaluated on cadence and on trigger; material change auto-opens a governance event.
Architecture Board	Bends	Keep for strategy, autonomy-graduation decisions, accountability assignment; add Trust & Safety competence; feed it continuous-assurance signals.
Architecture Contracts	Bends	Extend to specify autonomy level, guardrails in force, oversight obligations, and the named accountability owner for agent-performed work.
Compliance Reviews	Bends	Keep design-time review; pair with continuous compliance — ongoing automated checking against guardrails and eval baselines.

TOGAF GOVERNANCE CONSTRUCT	AEAF STATE	AEAF RESPONSE
— (no analogue)	Runtime governance loop (new)	Enforce guardrails in-line; monitor evals and drift; detect and contain incidents; adapt within bounds or escalate; route material change back into a new design-time cycle.

The design-time gate (amber — human judgment, the board, sign-off) and the runtime loop (indigo — machine-speed enforcement and monitoring) are joined at the handoff in Phase 9: the gate's last act is to approve the guardrails, autonomy level, eval thresholds, oversight mode, and accountability owner, and then hand the agent to the runtime loop.

23.3 The migration path for a TOGAF shop

A TOGAF practice does not adopt AEAF in one move, and it should not try. The path is incremental, and the order is chosen so that each step delivers value on work the practice is already doing before the next step is required. Figure 23.1 is the roadmap; the three steps below are its content.

The sequencing logic: **Step 1** is additive and low-risk — it extends models you already produce and introduces no new governance burden, so it can run inside your current ADM cycles. **Step 2** is needed the moment any work is probabilistic, because requirements-and-testing cannot hold agent behaviour. **Step 3** is needed the moment any agent operates with reduced human oversight, because a design-time gate alone cannot govern continuous behaviour. You do not start Step 3 until you have agents that earn it.

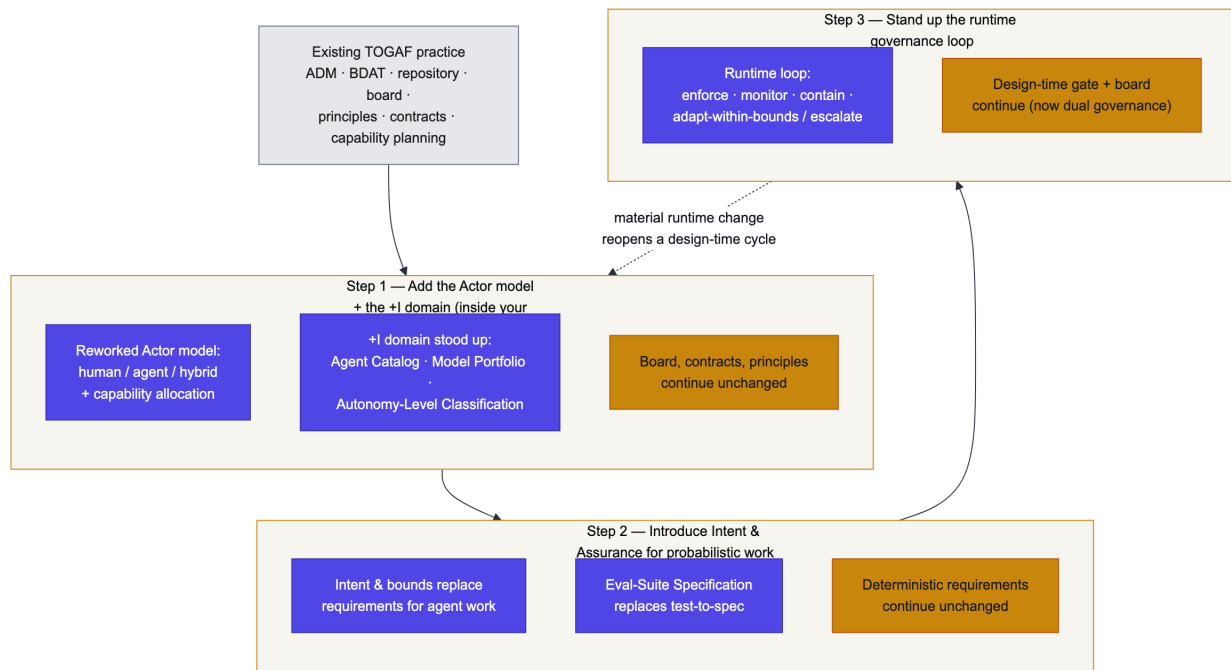


Figure 23.1 — The incremental migration roadmap. Indigo nodes are the new agentic additions a TOGAF shop introduces at each step; amber nodes are the human-governance constructs that continue unchanged throughout — the board, architecture contracts, the principle catalog, the design-time gate, and deterministic requirements. The grey base is the existing TOGAF practice that the whole path builds on rather than replaces. The dotted return edge is the dual-governance link: once the runtime loop exists, a material behavioural change reopens a design-time cycle rather than waiting for a periodic review.

23.3.1 Step 1 — Add the Actor model change and the +I domain

This is the entry move because it is purely additive and runs inside your current ADM cycles with no new governance burden. Two things go in.

- **The reworked Actor model.** The content metamodel's Actor is implicitly human. Introduce the human / agent / hybrid distinction so that an agent is a first-class actor with an autonomy level, an oversight mode, and a named accountable human owner (→ see AEAF Framework §7). In your Business Architecture work, this surfaces immediately as **capability allocation**: for each capability, record who or what performs it. You are already modelling capabilities; you now annotate them with the performer.
- **The +I domain.** Stand up Intelligence & Agency as a fifth domain and begin populating its foundational artifacts — at minimum the **Agent Catalog**, the **Model Portfolio**, and the **Autonomy-Level Classification** (→ see AEAF Framework §6). For a practice that has agents in production but no architectural home for them, this alone resolves the most

common gap: agents that exist but are not in the repository, not governed, and not owned.

Throughout Step 1, the board, contracts, principles, and the rest of the practice continue unchanged. Nothing is taken away.

23.3.2 Step 2 — Introduce Intent & Assurance for probabilistic work

The trigger for Step 2 is the first probabilistic capability — the first time an agent does work that requirements-and-testing cannot specify to a clean pass/fail. Correctness for such work is distributional, not binary, so "the requirement is met" stops being a true/false statement (→ see AEAF Framework §17).

For that work — and *only* that work — replace the pure requirement with **Intent & Assurance** at the centre:

- **Intent** declares the desired outcome and its bounds ("resolve refund requests within policy; never exceed €X without human approval; aim for $\geq 95\%$ customer-acceptable resolutions"), rather than enumerating steps.
- **Assurance** measures behaviour against an **Eval-Suite Specification** with pass thresholds, refreshed continuously rather than signed off once.

The discipline that keeps this honest for a TOGAF practice: **deterministic requirements continue unchanged**. A requirement is still a requirement for deterministic components. You are not abolishing Requirements Management; you are generalising the centre so that probabilistic work has a home that fits it, while deterministic work keeps the home it already has.

23.3.3 Step 3 — Stand up the runtime governance loop

The trigger for Step 3 is the first agent that operates with reduced human oversight — on-the-loop or out-of-the-loop rather than in-the-loop. Until an agent acts without a human approving each action, your design-time gate is sufficient. Once it does, the gate alone cannot govern behaviour that only exists at runtime.

Stand up the **runtime governance loop** alongside the gate you already run (→ see AEAF Framework §20): guardrails enforced in-line at decision points; eval suites and drift detectors monitoring behaviour continuously; incidents detected and contained; the agent adapting within its bounds or escalating to a human; and material change automatically reopening a design-time cycle. Your board and design-time gate do not go away — they become the design-time half of **dual governance**, and the gate gains one new responsibility: approving the handoff to the runtime loop.

This is the heaviest step, which is why it is last. It also carries the strongest **match-weight-to-consequence** rule: a low-consequence, reversible, in-the-loop agent needs light gating and may not need a heavy runtime loop at all; a high-consequence or out-of-the-loop agent needs the full design-time gate and a strong runtime loop (→ see AEAF Framework §9).

23.4 What to extend versus what to create new

The migration steps above translate directly into a worklist against your repository. Table 23.3 separates the two kinds of work so a practice can scope honestly: *extend* work reuses an existing asset and adds an attribute, axis, or view; *create-new* work has no predecessor and is genuinely additive.

Table 23.3 — Extend an existing TOGAF artifact, or create new. Most of the migration is extension. The create-new column is the honest measure of the genuinely additive effort.

EXTEND AN EXISTING ARTIFACT	WHAT YOU ADD TO IT
Capability model	Performer type (human/agent/hybrid) and autonomy level per capability.
Function/Application matrix	The Capability-Allocation expression and the agent-to-tool access view.
Data Entity Catalog	Knowledge corpora with provenance, freshness, and eligibility.
Application Portfolio Catalog	A pointer to the Agent Catalog; agents are not folded in as applications.
Principle catalog	An Enforcement attribute on every principle.
Standards Information Base	Approved models, eval-coverage standards, autonomy definitions, guardrail standards.
Architecture contracts	Autonomy level, guardrails in force, oversight obligations, accountability owner.
TIME portfolio assessment	An agentic-disposition axis (Augment / Agentify / Keep-Human / Retire).
Architecture Board composition	Trust & Safety competence; continuous-assurance signal feeds.

EXTEND AN EXISTING ARTIFACT	WHAT YOU ADD TO IT
Communication / process diagrams	Oversight/Escalation flow and autonomy-boundary views.
CREATE NEW (NO TOGAF PREDECESSOR)	WHY IT HAS NO ANALOGUE
Agent Catalog	Agents are a distinct actor type, neither application nor human.
Model Portfolio	No classical artifact tracks models, their providers, costs, and the agents they power.
Autonomy-Level Classification	TOGAF has no concept of graded independent action for a non-human actor.
Eval-Suite Specification	Evaluate-against-bounds has no test-to-spec predecessor.
Guardrail/Policy Catalog	Machine-enforceable, runtime-bounding policy is new.
Human-Agent RACI	TOGAF's RACI covers architecture work, not agent actions in production.
Trust & Accountability Matrix	Accountability for autonomous action in production has no analogue.
Capability Automation-Frontier Map	No classical artifact tracks the human/agent boundary or its trajectory.
The runtime governance loop	Design-time governance has no continuous, machine-speed half.

23.5 Running AEAF and TOGAF side by side during transition

A practice will not flip overnight, and it should not. For an extended period most of the portfolio is deterministic work governed exactly as before, and a growing minority is agentic work governed by the new machinery. The two run side by side, and the seam between them must be deliberate rather than accidental.

The rule that keeps the seam clean is **the determinism spectrum** (→ see AEAF Framework §4): a component is placed on a range from fully deterministic (same input, same output — specifiable and testable) to non-deterministic (probabilistic — bounded and evaluated). Where a component sits on that spectrum determines which machinery governs it, and the choice is recorded, not assumed.

CONCERN	DETERMINISTIC WORK	AGENTIC WORK
Centre	Requirements Management (unchanged TOGAF)	Intent & Assurance
Verification	Test-to-spec, pass/fail	Eval-against-bounds, distributional
Governance	Design-time gate only	Dual governance (gate + runtime loop)
Change	Episodic, change-request driven	Continuous assurance; auto- triggered cycles
Domain home	BDAT	BDAT, plus +I

Three practices make side-by-side work coherently rather than as two disconnected shops:

- 1. One repository, two readable layers.** Keep a single Architecture Repository. Add the machine-consumable runtime layer (principles, standards, guardrails published in a form agents query and obey at runtime) alongside the human-readable assets, versioned together (→ see AEAF Framework §8). You do not run two repositories.
- 2. One board, extended composition.** Keep one Architecture Board. Add Trust & Safety competence and feed it continuous-assurance signals; do not stand up a separate "AI board" that fragments accountability.
- 3. One capability model, allocated.** Keep one capability model for the whole enterprise. The human/agent boundary is an *allocation* recorded on that model (the Capability Automation-Frontier Map), not a fork into "the human architecture" and "the AI architecture."

The transition ends not when TOGAF is gone — deterministic work governed in the classical way is a permanent part of the landscape — but when the agentic machinery is established and the seam between the two is governed deliberately.

23.6 Common objections from a TOGAF practice, answered honestly

A mature practice will and should challenge a new framework. The objections below are the ones that recur; each gets a direct answer, not a deflection.

OBJECTION	HONEST ANSWER
"This is just TOGAF with AI bolted on. Why a new framework?"	Three things genuinely break, not bend: requirements cannot specify probabilistic behaviour; design-time governance cannot govern continuous post-deployment behaviour; and BDAT plus a human Actor have no home for agency. A bolt-on cannot fix a broken centre, a broken governance assumption, and a missing domain. The rest is extension — which is the point, and why most of TOGAF is kept.
"We just finished a TOGAF rollout. Is that wasted?"	No. Roughly three-quarters carries forward, mostly intact (→ see AEAf Framework §2). Your repository, principles, capability model, board, and contracts are the load-bearing structure AEAf builds on. The migration is largely extension of assets you already have.
"Our systems are deterministic. Does this even apply to us?"	For genuinely deterministic work, classical TOGAF governance is correct and AEAf changes nothing. AEAf applies the moment a probabilistic actor enters the workforce. If none has, adopting AEAf now is about being ready, not about reworking deterministic systems.
"AEAf isn't a recognised standard or trademarked like TOGAF."	Correct, and stated plainly: AEAf is independent original work, not affiliated with or endorsed by The Open Group, and not a TOGAF profile. It earns adoption on whether its analysis is right, not on a certification. It deliberately maps to TOGAF so an existing practice can judge it against what it already knows.

OBJECTION	HONEST ANSWER
"Runtime governance sounds like operations, not architecture."	Architecture owns the design of the runtime loop — the guardrails, autonomy levels, eval thresholds, oversight modes, and accountability owners that bound behaviour. Operations runs the loop; architecture governs what the loop is allowed to permit. The same division exists for security architecture today.
"Who is accountable when an agent acts? This feels legally unsettled."	AEAF's stance is firm: accountability never transfers to an agent. Every capability has exactly one named human accountability owner , recorded in the Human-Agent RACI and Trust & Accountability Matrix (→ see AEAF Framework §18). An agent can be responsible for execution; it is never accountable. That is a design rule precisely because the legal ground is unsettled.
"We can't staff a fifth domain."	Step 1 needs three foundational artifacts, not a standing organisation. The +I competency is added to existing architects' remit before it becomes anyone's full-time role; the role specialisation (agent owner, eval lead, AI-risk) is an adoption-archetype question, deferred to Book 2 (→ see AEAF in Practice §3 and §4).

23.7 Where this chapter stops

This chapter is the bridge: the four master correspondences, the three-step path, the extend-versus-create worklist, the side-by-side discipline, and the honest objections. It deliberately stops short of *how much* of AEAF a given enterprise should adopt and *in what configuration*.

Those are tailoring decisions, and they depend on what kind of enterprise is adopting — a startup running one copilot, a scale-up, a large enterprise, or a regulated institution each take a different subset and a different governance weight. AEAF treats tailoring as a first-class activity, exactly as TOGAF treats ADM tailoring, but the presets and archetypes belong to the adoption guide:

- **Tailoring AEAF** — select / modify / partially adopt; what you can drop and what you cannot; the phase-map applied to a real scope (→ see AEAF in Practice §2).
- **Adoption archetypes** — startup / scale-up / enterprise / regulated, each with a tailoring preset (→ see AEAF in Practice §3).

A TOGAF practice that has read this chapter knows the map, the order of moves, and the honest cost. Book 2 turns that into a plan sized for its own enterprise.

Appendices

In brief. These appendices are the quick-reference apparatus of Book 1. Appendix A is the complete AEAF glossary. Appendix B maps AEAF to TOGAF — phases, domains, key concepts, and artifacts — so an existing practitioner can see the correspondence at a glance. Appendix C gives ordered reading paths for five reader types. Appendix D is a one-page index of the AEAF principle catalog (the canonical catalog with rationale and implications is Chapter 8). Appendix E is a one-page meta-model reference with the core class diagram. Where a term is defined elsewhere in Book 1, these appendices restate the locked definition rather than introduce a new one.

TOGAF® is a registered trademark of The Open Group. AEAF is an independent framework and is neither derived from nor endorsed by The Open Group. AEAF answers, extends, and in places replaces TOGAF concepts; the mapping in Appendix B is a fair comparison for practitioners, not a claim of lineage, compliance, or affiliation.

Appendix A — Glossary

The locked AEAF lexicon, alphabetized. Each entry gives the term and its definition in one or two sentences. These definitions are canonical; a term means the same thing throughout both books. The home chapter for the full treatment is noted where it aids the reader.

TERM	DEFINITION
Accountability owner	The single named human or role answerable for the outcomes of a capability, including work performed by agent actors. Exactly one per capability; accountability never transfers to an agent.

TERM	DEFINITION
Action / tool	A concrete capability to affect the world that an agent may invoke — an API call, transaction, or message — carrying a permission, a budget or rate limit, and a reversibility attribute.
Agent (AI agent)	A software system that pursues goals by reasoning over a model, taking actions through tools, and operating within guardrails at a defined autonomy level. An agent is the engine plus tools, instructions, guardrails, and autonomy level — not the model alone.
Agent actor	An AI agent treated as a first-class actor in the architecture: it perceives, decides, and acts toward an objective with limited human direction, runs a model, draws on knowledge corpora, is bounded by guardrails, is assured by an eval suite, and is answerable via an accountability owner.
Agentic Enterprise	An enterprise in which AI agents act as first-class actors alongside humans — executing work, consuming architecture at runtime, and being governed as participants rather than tools. The AEAF target state.
Assurance	The continuing evidence — from evals, guardrail logs, and oversight — that an agent actor behaves within its intent and bounds over time. It is ongoing, not a one-time sign-off; it replaces acceptance testing for non-deterministic work.
Assurance case	The structured argument and evidence that an agent is fit to operate within its bounds, aggregating eval results and guardrails for an agent, with stated residual risk and a review date.
Automation frontier	The current boundary, per capability, between work performed by human actors and work performed by agent actors — and the direction that boundary is expected to move. A frontier moves; describe its position and trajectory.

TERM	DEFINITION
Autonomy level	The graded degree of independent action granted to an agent actor for a capability, from suggest-only through bounded action to full autonomy within guardrails, on the AEAf Autonomy-Level Classification scale.
BDAT+I	The five AEAf architecture domains: B usiness, D ata, A pplication, T echnology, plus I ntelligence & Agency. The "+I" is AEAf's defining structural addition to the four classical EA domains.
Blended workforce	The combined population of human actors and agent actors that performs the enterprise's work, allocated across the same capabilities. Not "hybrid workforce" — reserve "hybrid" for hybrid teams.
Capability	A stable, outcome-named ability of the enterprise to do something ("Settle a claim," "Onboard a supplier"), independent of who or what performs it. Capabilities are actor-agnostic by design, which is what lets work be reallocated between humans and agents.
Capability allocation	The decision and record of which actors — human, agent, or hybrid team — perform a given capability, at what autonomy level, under what guardrails, and under what oversight. The central AEAf design act.
Cognitive substrate	The body of data curated as material for machine cognition — knowledge corpora, embeddings and vector indexes, memory stores, and knowledge graphs — that grounds an agent's reasoning, as distinct from transactional data records.
Determinism spectrum	The range from fully deterministic system behaviour (same input, same output; specifiable and testable) to non-deterministic agent behaviour (probabilistic; bounded and evaluated rather than specified). Architecture spans the spectrum; components are placed on it, not sorted into a binary.

TERM	DEFINITION
Escalation	The defined path by which an agent actor hands control to a human (or higher-authority actor) when it hits a bound, low confidence, or a guardrail veto. An escalation is a designed path with a stated trigger, target, and fallback — not a failure.
Evaluation (eval) / eval suite	An evaluation is the measurement of an agent’s behaviour against expected outcomes on a defined set of cases. An eval suite is the maintained collection of such cases for a capability, with dimensions, thresholds, and cadence. Evals measure bounded behaviour; tests verify deterministic output.
Guardrail	A constraint that bounds an agent’s behaviour at runtime — input filter, output check, tool permission, spend limit, or action veto — enforced regardless of the agent’s reasoning. A guardrail bounds; it does not specify. One word, always.
Hybrid team	A unit of human and agent actors performing one capability together with shared context and defined handoff rules, jointly accountable for that capability.
Intelligence & Agency domain (+I)	The fifth AEAF architecture domain, covering models, agents, knowledge corpora, guardrails, evals, autonomy levels, and oversight — the domain that did not exist in BDAT. It is a domain, not a layer of the Application domain.
Intent	The bounded statement of what outcome an actor is to achieve and within what limits, replacing the fully specified requirement for non-deterministic work. Intent says what good looks like and the bounds; it does not enumerate steps. Pairs with assurance.

TERM	DEFINITION
Knowledge corpus	The governed body of content (documents, data, embeddings) that an agent retrieves from to ground its reasoning, with defined sources, freshness, classification, and access controls. Plural: corpora.
Model	The underlying machine-learning model (typically a large language or multimodal model) that provides an agent's reasoning and generation. One model can power many agents; the model is the engine, the agent is the engine in use.
Oversight modes	How a human relates to an agent's actions: in-the-loop (human approves each action before it executes), on-the-loop (human monitors and can intervene), and out-of-the-loop (no human in the runtime path; control is via guardrails and after-the-fact review).
Policy (machine-enforceable)	A rule expressed so that it can be enforced automatically at design time or runtime, not only read and obeyed by a human. The policy is the rule; a guardrail is one mechanism that enforces it.
Principle	A foundational architecture statement carrying a defined Enforcement attribute: advisory (guidance only), design-time (checked at a gate), or runtime (enforced continuously on agent behaviour). Every AEAF principle declares its enforcement.
Principle → guardrail compilation	The formal move by which a principle with enforcement = runtime is compiled into one or more machine-enforceable guardrails. It is the concrete expression of architecture having a second audience — agents at runtime, not only humans at design time.

TERM	DEFINITION
Runtime governance	Governance of an agent while it operates: guardrails enforced at decision points, behaviour monitored against eval baselines and drift detectors, incidents detected and contained, and material change escalated into design-time governance. The fast loop of the dual governance model.
The AEAF Method	The phased, cyclic method for developing and governing an Agentic Enterprise architecture — AEAF’s answer to the TOGAF ADM — with Intent & Assurance at the centre and a continuous runtime loop joined to the design-time cycle at governance.
Trust & Accountability	The cross-cutting AEAF concern for who is answerable for agent behaviour, behavioural risk and harm prevention, and accountable autonomy — treated as first-class and behaviour-focused, spanning all five domains, rather than as a sub-concern of security.

Appendix B — TOGAF ↔ AEAF mapping

These tables let an existing TOGAF practitioner locate the corresponding AEAF construct. The mapping is a navigation aid and a fair comparison; AEAF is independent original work, not a TOGAF profile or derivative (see the trademark and independence note at the head of these appendices). Where AEAF *bends* a construct it extends it; where it *breaks* a construct the assumption failed and AEAF replaces it; where a concern is *new* there is no TOGAF analogue.

B.1 — Method phases ↔ ADM phases

AEAF METHOD PHASE	TOGAF ADM ANALOGUE	STATUS	WHAT CHANGES
Preliminary	Preliminary	Bends	Stands up the Intelligence & Agency capability, dual-audience principle authoring, eval and guardrail infrastructure.
1 · Vision	A — Architecture Vision	Holds	Adds explicit blended-workforce intent and the autonomy/consequence-risk envelope to scope.
2 · Business & Capability	B — Business Architecture	Bends	Adds capability allocation (human/agent/hybrid) and the automation-frontier map as primary outputs.
3 · Data & Knowledge	C (data) — Data Architecture	Bends	Adds the cognitive substrate: knowledge corpora, embeddings, memory, lineage-for-accountability.
4 · Application	C (app) — Application Architecture	Bends	Designs for non-determinism: bounded outputs, fallback, human handoff, observability; Agent and Model as component types.

AEAF METHOD PHASE	TOGAF ADM ANALOGUE	STATUS	WHAT CHANGES
5 • Technology	D — Technology Architecture	Bends	Adds the agentic technology stratum (inference, vector stores, eval/observability, guardrail enforcement) and agent-aware security.
6 • Intelligence & Agency	— (no analogue)	New	The whole phase is new: models, agents, orchestration, knowledge use, guardrails, evals, autonomy levels.
7 • Opportunities & Solutions	E — Opportunities & Solutions	Holds	Adds agent-rollout, eval-suite build, and autonomy-graduation work-package types; sequences by trust earned.
8 • Migration Planning	F — Migration Planning	Bends	Adds autonomy-graduation gates and explicit planning of frontier moves and the human transition.
9 • Implementation Governance	G — Implementation Governance	Breaks	The design-time gate now also approves guardrails, autonomy levels, eval thresholds, oversight mode, and accountability owner, then hands off to the runtime loop.

AEAF METHOD PHASE	TOGAF ADM ANALOGUE	STATUS	WHAT CHANGES
10 · Change & Runtime Assurance	H — Architecture Change Management	Breaks	Replaces episodic change with the continuous runtime loop; material behavioural change auto-opens a governance event.
Intent & Assurance Management (centre)	Requirements Management (centre)	Breaks	Probabilistic work is bounded and evaluated, not specified and tested; deterministic requirements still apply unchanged.

B.2 — Domains: BDAT ↔ BDAT+I

TOGAF DOMAIN	AEAF DOMAIN	STATUS	WHAT CHANGES
Business (B)	Business	Holds, extended	Capability allocation and the automation frontier become first-class; roles gain performer-type.
Data (D)	Data	Holds, extended	Adds knowledge corpora, embeddings, and lineage-for-accountability alongside transactional data.
Application (A)	Application	Holds, extended	Adds Agent and Model as first-class component types; runtime, non-deterministic behaviour is modelled.

TOGAF DOMAIN	AEAF DOMAIN	STATUS	WHAT CHANGES
Technology (T)	Technology	Holds, extended	Adds the agentic stratum: inference, model gateways, vector stores, eval pipelines, guardrail enforcement points.
— (no analogue)	Intelligence & Agency (I)	New	Owns models, agents, knowledge corpora, guardrails, evals, autonomy levels, capability allocation, and the human-agent collaboration model.

B.3 — Key concepts

TOGAF CONCEPT	AEAF CONCEPT	STATUS	WHAT CHANGES
Requirements Management	Intent & Assurance	Breaks	Declare intended behaviour and bounds, then evaluate against eval suites and thresholds rather than test against a fixed spec.
Actor (implicitly human)	Actor model (human / agent / hybrid)	Breaks	Actor becomes abstract and specializes into human, agent, and hybrid team; the blended workforce is first-class.

TOGAF CONCEPT	AEAF CONCEPT	STATUS	WHAT CHANGES
Architecture Principles (Name / Statement / Rationale / Implications)	Dual-audience principles	Bends	The four-part structure holds; AEAF adds the Enforcement attribute (advisory / design-time / runtime) and a machine-consumable form for runtime principles.
Architecture Governance (single-pass, pre-deployment)	Dual governance	Breaks	Design-time gates are kept and paired with runtime enforcement plus continuous assurance, with automated escalation into human governance bodies.
Architecture Contracts	Extended contracts	Bends	Specify autonomy level, guardrails in force, oversight and escalation obligations, and the named accountability owner for agent work.
TIME portfolio model	TIME + agentic-disposition axis	Bends	Adds an Augment / Agentify / Keep-Human / Retire axis alongside Tolerate / Invest / Migrate / Eliminate.
Verification by test-to-spec	Evaluation against bounds	Breaks	Probabilistic components are assured by eval suites and thresholds, not pass/fail tests.

TOGAF CONCEPT	AEAF CONCEPT	STATUS	WHAT CHANGES
Architecture Board	Board + Trust & Safety competence	Bends	Kept for strategy, autonomy-graduation, and accountability assignment; gains Trust & Safety competence and continuous-assurance signals.

B.4 — Artifacts

TOGAF ARTIFACT (CATALOG / MATRIX / DIAGRAM)	AEAF ARTIFACT	STATUS
Application / Technology / Data / Principles catalogs	Retained, extended	Bends
Function/Application and Interaction matrices	Retained, extended	Bends
Communication / Process / Infrastructure diagrams	Retained, extended; plus oversight/escalation and autonomy-boundary views	Bends
—	Agent Catalog — every agent actor: purpose, model, tools, autonomy level, guardrails, accountability owner	New
—	Model Portfolio — models in use, providers, capabilities, costs, risk profile, and the agents each powers (TIME-equivalent)	New
—	Knowledge-Corpus Catalog — each corpus: sources, freshness, access controls, and the agents that retrieve from it	New

TOGAF ARTIFACT (CATALOG / MATRIX / DIAGRAM)	AEAF ARTIFACT	STATUS
—	Guardrail/Policy Catalog — machine-enforceable policies and the guardrails that enforce them, mapped to enforcement point	New
—	Human-Agent RACI / Decision-Authority Matrix — responsibility and accountability split between human and agent actors per capability	New
—	Autonomy-Level Classification — the autonomy-level scale and each capability's assigned level	New
—	Eval-Suite Specification — eval cases, expected outcomes, and pass thresholds per agent capability	New
—	Trust & Accountability Matrix — capabilities mapped to accountability owners, oversight modes, and assurance evidence	New
—	Capability Automation-Frontier Map — the current and projected human/agent boundary across the capability model	New

The nine new artifacts are defined in Book 1 Ch.21 and templated with worked examples in Book 2 Ch.15.

Appendix C — Reading paths

Book 1 is a reference; few readers read it cover to cover on the first pass. The paths below give an ordered subset of Book 1 chapters for five reader types. Chapter numbers follow the Book 1 table of contents.

READER	GOAL	ORDERED READING PATH
Executive / sponsor	Understand the stakes, the target state, and what adoption commits the enterprise to — without the method detail.	1 (the inflection) → 3 (the Agentic Enterprise) → 7 (blended workforce) → 18 (trust, safety & accountability) → 22 (maturity model) → 23 (adopting alongside TOGAF).
Enterprise architect	Build a working command of the framework end to end.	2 (what TOGAF gets right, where it stops) → 5 (meta-model) → 6 (BDAT+I) → 7 (Actor model) → 8 (principles) → 9 (the Method) → 11 (business & capability) → 17 (Intent & Assurance) → 21 (artifact catalog) → 23 (migration).
AI / solution architect	Design the model/agent/ knowledge/guardrail stack and how it behaves at runtime.	4 (core concepts) → 12 (data & knowledge) → 13 (application in a non-deterministic world) → 14 (technology / AI platform) → 15 (Intelligence & Agency architecture) → 17 (Intent & Assurance) → 20 (continuous & runtime governance).
Risk / governance lead	Understand accountability, oversight, and how agentic systems are governed in operation.	3 (the four breaks) → 8 (principles & the Enforcement attribute) → 17 (Intent & Assurance) → 18 (trust, safety & accountability) → 19 (human-agent operating model) → 20 (continuous & runtime governance).

READER	GOAL	ORDERED READING PATH
Existing-TOGAF practitioner	Map what they know to AEAf and adopt without discarding their practice.	2 (fair appraisal + gap analysis) → 23 (adopting alongside / migrating from TOGAF) → 5 (meta-model changes) → 9 (Method vs ADM) → 8 (the one added principle attribute) → Appendix B (the full mapping).

Appendix D — The AEAf principle catalog (index)

This is a quick-reference index, not the catalog. The canonical AEAf principle catalog — each principle with its full statement, rationale, and implications — is → **see AEAf Framework §8 (Principles for the Agentic Enterprise)**. Chapter 8 is authoritative; where this index and Chapter 8 differ, Chapter 8 governs.

Every AEAf principle carries an **Enforcement** attribute — **advisory** (human judgement), **design-time** (checked at a review gate), or **runtime** (compiled into a guardrail and enforced continuously). The principles are organized into five families. The entries below are representative of each family; the complete numbered catalog and the precise enforcement value per principle are fixed in Chapter 8.

Foundational

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Capability is actor-agnostic	Capabilities are defined by outcome, independent of whether a human, agent, or hybrid team performs them.	Design-time
Allocation is a deliberate decision	Who or what performs a capability — and at what autonomy level — is decided and recorded, never defaulted.	Design-time

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Architecture has a second audience	Architecture assets that bind agents are published in a machine-consumable form, not prose alone.	Design-time

Safety & trust

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Accountability never transfers to an agent	Every agent-performed capability has exactly one named, answerable human owner.	Design-time
Bound before you grant autonomy	An agent operates only within guardrails approved for its autonomy level.	Runtime
Escalation is designed, not improvised	Every agent has a defined path to hand control to a human on a bound, low confidence, or veto.	Runtime

Assurance

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Evaluate, do not only specify	Non-deterministic behaviour is assured against eval suites and thresholds, not tested to a fixed spec.	Design-time
Assurance is continuous	Evidence that an agent stays within intent is refreshed on cadence and on trigger, not signed off once.	Runtime
Material change reopens governance	A model swap, corpus change, or failed assurance case auto-opens a governance event.	Runtime

Workforce

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Match oversight to consequence	Oversight mode (in / on / out-of-the-loop) is set by the consequence and reversibility of the work.	Design-time
Plan the human transition	Frontier moves account explicitly for reskilling and role change, not only re-allocation.	Advisory

Architecture

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Model, agent, and application are distinct	The reasoning model, the agent that uses it, and the runtime that hosts it are modelled separately.	Design-time
Knowledge is governed substrate	Every knowledge corpus has defined sources, freshness, classification, and access control.	Design-time
Dual governance is the standing model	Design-time gates and a continuous runtime loop govern agentic systems together, not in isolation.	Runtime
Technology & vendor independence (R4)	The architecture commits to capabilities and option-classes; any vendor or product named is illustrative, and lock-in is a recorded trade-off, never a default.	Advisory + design-time

PRINCIPLE (REPRESENTATIVE)	ONE-LINE STATEMENT	ENFORCEMENT
Principle-led, protocol-pluggable (R5)	Connecting protocols are adopted as guardrailed actions under an autonomy level and accountability owner, never as framework dependencies; AEAF commits to principles and option-classes, not to any single protocol.	Advisory + design-time

The catalog comprises eighteen principles across these five families. The exact list, the full four-part structure (Name / Statement / Rationale / Implications + Enforcement), and the canonical enforcement value for each are defined in Chapter 8.

Appendix E — Meta-model quick reference

One-page entity reference. The full treatment is → see **AEAF Framework §5 (The AEAF meta-model)**. Status legend: **New** (no TOGAF analogue) · **Extended** (TOGAF entity with new attributes or semantics) · **Retained** (used as in TOGAF). The single most important change is that **Actor is no longer synonymous with human**.

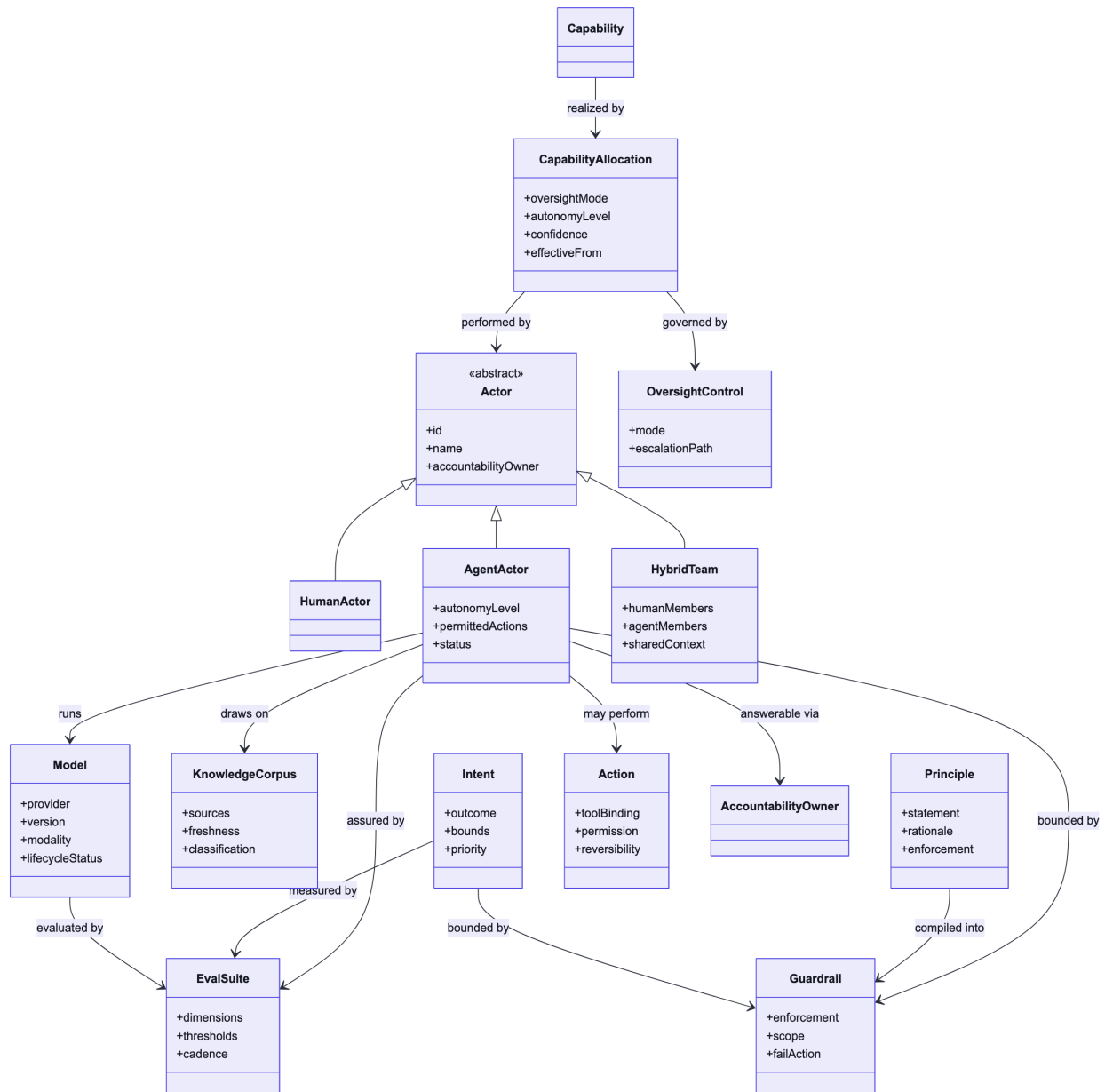
ENTITY	STATUS	ONE-LINE DEFINITION
Actor	Extended	Anything that performs capabilities and bears responsibilities; abstract, no longer implying human.
Human Actor	Extended	A person (or organizational role filled by people) performing work.
Agent Actor	New	A model-driven actor that perceives, decides, and acts toward an objective with limited human direction.

ENTITY	STATUS	ONE-LINE DEFINITION
Hybrid Team	New	A unit of human and agent actors performing a capability together with shared context and handoff rules.
Model	New	The underlying ML/LLM (or ensemble) that powers an agent or AI service; distinct from the agent that wraps it.
Knowledge Corpus	New	Data curated as substrate for machine cognition — RAG corpora, embeddings, memory, knowledge graphs.
Capability	Retained	A business ability the enterprise has or needs.
Capability Allocation	New	The decision of who/what performs a capability and under what oversight — the central human-agent join.
Automation Frontier	New	The derived, time-versioned boundary between human- and agent-performed work across the capability map.
Autonomy Level	New	A classification of how much independent action an agent is granted, from assistive to autonomous-within-bounds.
Principle	Extended	A normative rule guiding decisions; now dual-audience, with an Enforcement attribute.
Guardrail / Policy	New	A machine-enforceable constraint on agent behaviour, derived from a principle, standard, or regulation.

ENTITY	STATUS	ONE-LINE DEFINITION
Intent	New	A declared desired outcome with bounds and priority — replaces the fully specified requirement for probabilistic work.
Eval Suite / Evaluation	New	The tests, probes, and metrics that bound and monitor a model's or agent's behaviour.
Assurance Case	New	The structured argument and evidence that an agent is fit to operate within its bounds.
Oversight Control	New	How humans supervise an allocation: mode (in / on / out-of-the-loop) and escalation path.
Action / Tool	New	A concrete capability to affect the world that an agent may invoke, with permission and reversibility.
Accountability Owner	New	The named human or legal entity answerable for an agent's actions and outcomes.
Role / Function / Process / Service	Extended	As in TOGAF, but a role may now be filled by a human, an agent, or a hybrid team.
Application Component	Extended	As in TOGAF; an AI service or agent runtime is a component, but the Agent it hosts is modelled separately.
Data Entity	Extended	As in TOGAF; gains the notion that data may be curated into a Knowledge Corpus.

ENTITY	STATUS	ONE-LINE DEFINITION
Technology Component	Extended	As in TOGAF; now includes inference platforms, vector stores, orchestration/runtime, model gateways.
Driver / Goal / Objective	Retained	As in TOGAF; traced by Intent.
Building Block (ABB/SBB)	Retained	As in TOGAF; AEAF adds agentic reference building blocks.

Figure E.1 — The AEAF meta-model core. Indigo-side entities (Agent, Model, Guardrail, Eval Suite) are the machine-intelligence additions; Capability Allocation and Oversight Control are the human-agent join points; Accountability Owner anchors every agent to a human/legal answerability.



The traceability chains the meta-model preserves – Driver/Goal → Intent → Eval Suite → Assurance Case; Principle → Guardrail → enforcement point; Capability → Allocation → Actor + Oversight + Autonomy Level → Accountability Owner; Model/Agent → Eval Suite → Runtime Assurance → Incident → corrective change – are detailed in → see AEAF Framework §5 and underpin the continuous-governance loop of the Method (→ see AEAF Framework §9).