AEA by Ivory & Co.

Execution Architecture Standard

Version: v1.0.0 — Published Anchor: Instructions

1. Definition

> Execution Architecture defines the enterprise's instructions for how work is performed.

Execution Architecture is the architecture of Instructions.

Its governing question is:

> How is this work to be performed?

Execution translates required enterprise work into bounded, reusable and executable instruction structures without assuming that every context requires an entirely different process.

Its native architecture is:

> Process Module !’ Canonical Invariant Contract !’ Approved Variant(s) !’ Task(s) !’ Instruction(s)

Operational Policies, Operational RACI and Operational Metrics govern or qualify the performance of that work.

2. Why Execution Architecture Exists

A Capability establishes what the enterprise is able to do.

It does not describe how work is performed.

A Role establishes participation.

It does not describe the work itself.

An Environment enables work.

It does not provide the instructions.

A Relationship may establish an obligation.

It does not define the executable mechanism for satisfying it.

Execution Architecture exists to make the enterprise's work explicit, reusable, governable and executable.

It enables work to remain coherent while allowing legitimate contextual variation.

> Execution Architecture is modular, collapsible and composable.

3. Process Module

A Process Module is a bounded, reusable unit of enterprise work whose identity is established by a Canonical Invariant Contract.

A Process Module represents work at a level where it can be:

- understood as a coherent unit;
- reused where appropriate;
- composed with other Process Modules;

- varied without losing identity;
- assigned participation requirements;
- measured;
- and governed.

A Process Module is not defined by a fixed diagram depth or universal hierarchy.

Its boundary is determined by the invariant contract that gives the work its identity.

4. Canonical Invariant Contract

The Canonical Invariant Contract establishes what must remain true for a Process Module to remain the same Process Module.

At minimum it contains:

1. Trigger 2. Inputs 3. Bounded Purpose 4. Output 5. Completion Meaning 6. Evidence 7. Exceptions

These are the invariants of process identity.

If a proposed Variant changes the core Trigger, Bounded Purpose, Output, Completion Meaning or Essential Evidence, it is not merely a Variant. It is a different Process Module or module proposition and must be retested accordingly.

This prevents materially different work from being hidden beneath one process name.

5. Trigger

A Trigger is the condition or event that initiates the Process Module.

The Trigger helps establish where the work begins.

A contextual difference in how a Trigger is detected or routed may be a Variant where the underlying initiating condition remains materially the same.

A materially different initiating proposition may indicate a different Process Module.

6. Inputs

Inputs are the materials, information, conditions, requests, items or other propositions required for the Process Module to begin or proceed.

Inputs establish the inbound interface of the module.

An Input may originate from another Process Module, an Ecosystem Exchange, an Environment, a Participant, a system, a customer interaction or another relevant source.

The source architecture retains ownership of the originating object.

Execution owns its use as an Input to the work.

7. Bounded Purpose

The Bounded Purpose establishes why the Process Module exists as a unit of work.

It must be sufficiently precise to distinguish the module from adjacent work without turning the Process Module into a restatement of an entire Capability or enterprise outcome.

The Bounded Purpose helps determine whether contextual differences remain Variants or constitute different work.

8. Output

An Output is what the Process Module produces when performed.

The Output forms part of the module's identity and its interface with subsequent work or other architectural propositions.

An Output may become an Input to another Process Module.

An Output may also contribute to a broader operational outcome.

Output identity must not be silently changed within an Approved Variant.

9. Completion Meaning

Completion Meaning defines what it means for the Process Module to be complete.

Completion is not merely the final step having been attempted.

It establishes the condition that must be satisfied for the module to be considered finished.

This is essential for human and agent execution because a sequence can be exhausted without the required work actually being complete.

> Completion is a condition, not simply the end of a list.

10. Evidence

Evidence within the Canonical Invariant Contract identifies what must exist to demonstrate that the Process Module has been performed or completed as required.

Execution establishes the evidence requirement associated with the work.

Intelligence Architecture establishes the information, provenance, source and interpretation architecture required where that evidence must be understood.

Reality may capture the actual evidence state or performance actuality.

> Execution defines what evidence the work requires. Intelligence enables that evidence to be understood. Reality captures what actually exists.

Essential Evidence forms part of Process Module identity where removing or materially changing it would alter the meaning of completion.

11. Exceptions

Exceptions establish conditions under which normal execution cannot or should not proceed as intended.

An Exception may:

- stop work;
- route work differently;
- require escalation;
- invoke another Process Module;
- require human intervention;
- trigger review;
- or create another governed response.

Exception handling remains part of Execution where it specifies what work must occur.

The condition causing the Exception may originate in another architecture.

12. Approved Variant

An Approved Variant is a governed contextual difference in the execution of a Process Module that preserves the Process Module's Canonical Invariant Contract.

A Variant may alter:

- timing;
- sequence;
- channel;
- thresholds;
- controls;
- participants;
- runtime;
- task arrangement;
- instructions;
- or other contextual execution details.

A difference elsewhere in AEA does not automatically create an Execution Variant.

The governing test is:

> Does the contextual difference materially change the executable instructions while preserving the Process Module's invariant identity?

If no, there may be no Execution Variant.

If yes, an Approved Variant may be required.

If the invariant identity changes, a different Process Module is required.

13. Variant Causation

The cause of a Variant may originate outside Execution.

Examples include:

- an Environment constraint;
- a Workforce Participant Type;
- an Ecosystem Relationship condition;
- a Trust Reliance Condition;
- an Expression Audience or experience requirement;
- a regulatory obligation;
- or another architectural condition.

The cause remains owned by its source architecture.

Execution owns the resulting change to executable work.

> Source condition !’ Execution impact !’ Approved Variant where required.

Cross-architecture contribution does not transfer ownership.

14. Task

A Task is a bounded unit of work within a Process Module or Approved Variant.

Tasks provide useful decomposition where the work needs greater executable resolution.

There is no mandatory universal Task depth.

A Process Module may contain several Tasks.

A simple Process Module may require little or no explicit Task hierarchy where Instructions can be represented directly without loss of clarity.

15. Instruction

An Instruction defines the executable detail required to perform work within a Process Module, Approved Variant or Task.

Instructions are the most direct expression of the Execution anchor.

They may be used by:

· humans;
· agents;
· systems;
· organisations;
· or hybrid fulfilment arrangements,
where appropriate.

The form of an Instruction may differ by fulfiller or runtime while preserving the required work.

There is no universal Instruction depth.

The required level is the level at which the work can be performed reliably and governed appropriately.

16. Operational Policy

An Operational Policy is a rule governing how the enterprise's instructions must be applied.

Operational Policy belongs to Execution where the rule governs the performance of work.

The source obligation may originate elsewhere.

For example:

· a contractual commitment may originate in Ecosystem;
· a regulatory obligation may arise through an Ecosystem Relationship;
· a Trust condition may constrain execution;
· a Directional guardrail may constrain permissible action.
Execution may translate that obligation into an Operational Policy without acquiring ownership of the source proposition.

Source obligation "` operational application rule.

Traceability between them must be preserved where material.

17. Operational RACI

Operational RACI belongs to Execution Architecture.

It expresses participation and accountability requirements within the work.

Operational RACI may establish who or what participation is required to be:

· Responsible;

· Accountable;
· Consulted;
· Informed;

or another governed participation designation where established.

Workforce Architecture consumes those participation requirements when composing Roles and assigning Participants.

> Execution defines the participation required by the work. Workforce defines how that participation is composed and fulfilled.

Operational RACI does not by itself constitute the Workforce Authority Structure.

> Operational RACI "` Authority Structure.

An Accountable designation within work must not be interpreted as an unlimited grant of binding enterprise decision authority.

Workforce Authority Structure establishes binding decision rights and delegation.

18. Operational Metrics

Operational Metrics belong to Execution Architecture.

Their governing question is:

> Is the work being performed as it is intended to be performed?

Execution defines the metric and the intended performance proposition.

Reality captures the actual observed or derived value.

Intelligence may establish the information, evidence, provenance, context or derivation required to understand the metric.

The governing separation is:

> Execution defines the measure !’ Intelligence may establish the derivation !’ Reality captures the actual value.

Measurement ownership and measured actuality are different propositions.

Operational Metrics must not be confused with Domain measures of whether a Capability is producing the outcome for which it exists.

19. Modular, Collapsible and Composable

Execution Architecture is modular because work is represented through bounded Process Modules.

It is collapsible because different levels of executable detail can be exposed without changing the identity of the underlying work.

It is composable because Process Modules can connect through their interfaces to form larger operational mechanisms.

Composition does not require a separate universal inter-module object in the Published Standard.

The Process Module's Trigger, Inputs, Outputs, Completion Meaning and other interfaces provide the basis for composition.

> Do not invent additional architecture where the module interfaces already establish the connection.

20. Process Interfaces

Process Modules naturally connect through their boundaries.

A useful interface lens may consider:

- Supplier;
- Input;
- Process;
- Output;
- Customer;

where relevant.

This SIPOC-style view may help identify the interface between modules without becoming a mandatory Execution object taxonomy.

The source of an Input or recipient of an Output may be an Ecosystem Actor, Workforce Participant, another Process Module or another architectural proposition.

Execution preserves the work interface without taking ownership of the originating or receiving object.

21. Execution Coverage

Execution Architecture does not have an intrinsic universal completeness threshold.

The enterprise does not need a process merely because a process could be documented.

Domain Architecture provides a primary coverage landscape by establishing the Capabilities through which the enterprise's value logic operates.

The governing test is:

> Where does the enterprise require explicit instructions in order for its Capabilities and intended outcomes to be realised reliably?

This does not create a one-to-one relationship between Capability and Process Module.

One Capability may require many Process Modules.

One Process Module may support several Capabilities.

Some work may require greater instructional resolution than other work.

22. Outcomes and Execution

Execution Architecture is concerned with operational outcomes insofar as they establish what work must produce and what completion means.

This does not transfer ownership of Directional Ultimate Outcome to Execution.

Directional owns the Ultimate Outcome the enterprise ultimately intends to cause externally.

Execution establishes the bounded operational outputs and completion conditions through which work contributes to enterprise outcomes.

> Strategic outcome intent and operational completion are different propositions.

23. Lessons Learned and Looping

Execution Architecture should support deliberate learning from work.

Where lessons learned reveal that:

- an Instruction is ineffective;
- a Variant is repeatedly required;
- an Exception is becoming normal;
- a completion condition is inadequate;
- evidence is insufficient;
- or a Process Module no longer represents the work correctly,

the relevant Execution proposition should be reviewed.

Learning does not mean silently changing instructions at runtime.

Lessons must be captured and routed through the appropriate governance and architectural change mechanism.

Where appropriate, processes may deliberately include learning or feedback loops as part of their executable design.

> Execution should be capable of learning without allowing uncontrolled drift.

24. Centre of Gravity

The design and application of enterprise instructions should reflect the enterprise's established centre of gravity where that concept is used in AEA application.

This may affect where:

- judgement sits;
- control is concentrated;
- exceptions are escalated;
- autonomy is permitted;
- work is standardised;
- or variation is tolerated.

Centre of Gravity does not override architecture ownership.

It is an application lens affecting how Execution is designed and composed with the wider Architecture Set.

Detailed Centre of Gravity application belongs in AEA in Action.

25. BOAS

A Business Operations Assembly System (BOAS) is an interlocking system of the business's work assembled from relevant architectural propositions across AEA into a coherent operational mechanism.

BOAS is not a native Execution Architecture object.

Execution contributes Process Modules, Variants, Tasks, Instructions, Operational Policies, Operational RACI and Operational Metrics.

Other architectures contribute their relevant propositions.

Intelligence enables the enterprise to establish the understanding required to bring those

propositions together correctly.

AEA in Action performs the cross-architecture composition.

> Intelligence enables assembled understanding. AEA in Action constructs the BOAS render.

BOAS must not be assigned to Intelligence merely because Intelligence enables the necessary understanding.

It must not be assigned to Execution merely because Execution contributes much of the work content.

26. SOP

A Standard Operating Procedure (SOP) is a bounded operational render subordinate within the BOAS hierarchy that describes how a relevant part of the BOAS is delivered.

SOP is not a native Execution Architecture object.

Execution provides much of the instruction content.

The SOP may also require propositions from Workforce, Environment, Trust, Ecosystem, Intelligence or other architectures.

The hierarchy is:

> BOAS !’ SOP

The governing coverage lens is:

> BOAS — Outcome
>
> SOP — Instructions
>
> User Journey — Experience

These are purpose-specific cross-architecture renders.

They are not peer native objects inside Execution Architecture.

27. User Journey

A User Journey is a cross-architecture render used to understand how a relevant user or audience encounters the enterprise across a sequence of interactions or experiences.

The user is not assumed to be a customer.

Depending upon the use case, the relevant user may be:

- a customer;
- employee;
- partner;
- supplier;
- regulator;
- intermediary;
- or another relevant Actor or Participant.

Expression contributes intended identity, Audience, Audience Segment and Experience propositions.

Execution contributes the work encountered through the journey.

Reality may contribute actual experience.

Ecosystem may contribute Actor and Relationship context.

Workforce may contribute participation.

Environment may contribute physical and technological enablement.

Intelligence enables the understanding required to assemble the view.

AEA in Action constructs the User Journey render.

> User Journey tests experience across the architecture. It is not a native Execution object.

28. BOAS, SOP and User Journey Coverage

Together, BOAS, SOP and User Journey provide different lenses over operational completeness:

> BOAS tests the outcome.
>
> SOP tests the instructions.
>
> User Journey tests the experience.

These lenses may expose gaps in Execution.

They do not establish a separate architecture.

Their composition methods belong in AEA in Action.

29. Execution and Directional Architecture

Directional defines the strategic scope and choices the business must be designed to realise.

Execution defines how work is performed within that direction.

Directional may establish constraints, priorities or guardrails that affect Execution.

Execution may operationalise those propositions through Process Modules, Operational Policies or other instruction structures.

Direction does not itself provide executable instructions.

Direction also does not confer Authority.

> Directional establishes strategic intent and choice. Execution establishes work. Workforce Authority Structure establishes binding decision rights.

30. Execution and Domain Architecture

Domain owns Business Models, Value Chains and Capabilities.

Execution owns the instructions through which work is performed.

A Capability is not a Process.

A Process is not a Capability.

Domain establishes the stable ability the enterprise requires.

Execution establishes how work is performed in support of that ability.

> Domain is the enterprise's capability landscape. Execution is the enterprise's instruction architecture.

Capability identity must not be inferred from current process design.

Process redesign does not automatically change Capability identity.

31. Execution and Ecosystem Architecture

Ecosystem establishes Actors, Relationships, Exchanges, Dependencies and relevant Relationship Commitments.

Execution establishes the work used to satisfy or operate within those relational conditions.

A contractual commitment is not an Instruction.

A Relationship is not a Process.

Where an Ecosystem condition materially changes executable work, Execution assesses whether an Operational Policy, Approved Variant or different Process Module is required.

> Ecosystem establishes the relationship condition. Execution establishes the work response.

32. Execution and Environment Architecture

Execution establishes the instructions.

Environment establishes the physical and technological enablement required for those instructions to be executable as intended, and the fulfilment of those enablement requirements.

The governing handoff is:

> Execution instruction !’ Environment Requirement !’ Requirement Fulfilment

A change in technology does not automatically create a new Process Module.

Where the enablement change materially changes executable work while preserving the Canonical Invariant Contract, an Approved Variant may be required.

Where executable work does not materially change, no Execution Variant is required merely because the Environment changed.

> Execution owns the instructions. Environment owns physical and technological enablement.

33. Execution and Workforce Architecture

Execution establishes participation requirements through the work.

Workforce composes those requirements into Roles and determines sourcing, Participant Type, Structures and assignment.

> Execution Participation Requirement !’ Workforce Role

Execution does not begin with named Participants.

Workforce does not own the Process merely because a Role participates in it.

Operational RACI remains Execution.

Authority Structure remains Workforce.

> Operational accountability within work and binding decision authority are distinct propositions.

34. Execution and Intelligence Architecture

Execution originates Requirements of Understanding where it needs to understand:

· work performance;

· evidence;
· exceptions;
· process behaviour;
· operational metrics;
· execution patterns;
· or another Execution proposition.

Intelligence architects fulfilment of those requirements.

Execution defines what evidence the work requires.

Intelligence establishes how that evidence can be sourced, interpreted, contextualised and understood.

Reality captures actual operational values and events.

Execution determines what it needs to understand. Intelligence establishes how that understanding is formed.

Intelligence does not own the Process because it can interpret it.

35. Execution and Possibility Architecture

Possibility may explore alternative processes, operating configurations, automation patterns or future ways of performing work.

Those remain hypotheses or possibilities until adopted.

Execution owns established instructions.

A possible future process does not become an Execution Process Module merely because it has been modelled.

Possibility explores how work could be performed. Execution defines how adopted work is to be performed.

36. Execution and Reality Architecture

Execution defines intended work and Operational Metrics.

Reality captures what actually occurs.

Reality may capture:

· actual process execution;
· actual completion;
· actual exceptions;
· actual evidence;
· actual operational metric values;
· actual timing;
· actual variation;
· or another material execution state.

Reality does not acquire ownership of the Process Module.

Execution does not own actual performance merely because it defined the intended performance.

Execution defines intended execution. Reality captures actual execution.

37. Execution and Trust Architecture

Trust determines warranted Reliance between parties under defined conditions.

Trust conditions may materially affect execution.

For example, a Trust proposition may require:

- additional review;
- reduced autonomy;
- different approval;
- human intervention;
- altered transaction limits;
- additional evidence;
- or another execution response.

Trust owns the Reliance Condition and Trust consequence.

Execution owns the resulting executable instruction change.

A Trust difference does not automatically create an Approved Variant.

The governing test remains whether executable work materially differs while preserving the Canonical Invariant Contract.

Trust determines warranted Reliance. Execution determines the work response where reliance conditions affect work.

38. Execution and Expression Architecture

Expression owns Enterprise Identity, Audience Segmentation, contextual Expression and intended Experience.

Execution owns the work through which some of that expression is delivered.

An Audience difference does not automatically create an Execution Variant.

An Expression difference becomes an Execution concern only where executable work materially changes.

Likewise, a service experience requirement may create an Execution requirement without transferring ownership of intended Experience to Execution.

Expression defines how the enterprise should be encountered. Execution defines how relevant work is performed.

User Journey remains a cross-architecture render.

39. Execution and AEA Governance

Execution Architecture defines enterprise instructions.

AEA Governance controls the authority, version, change, conformance, release and archival treatment of governed AEA material.

Operational change to work and change to the AEA Standard are not the same proposition.

Where Process Modules, Variants, Operational Policies or other governed Execution artefacts are subject to enterprise change control, the applicable governance mechanism must be followed.

Lessons learned and runtime adaptation must not silently bypass governed change.

40. Execution and AEA in Action

AEA in Action applies Execution Architecture together with other AEA architectures for specific enterprise purposes.

It may construct:

· BOAS;
· SOPs;
· User Journeys;
· service blueprints;
· operational diagnostics;
· cross-architecture operating views;
· change impact views;
· or other purpose-specific renders.

Execution contributes the instruction architecture.

Intelligence enables the understanding necessary for composition.

AEA in Action performs the composition.

Execution defines the work. AEA in Action assembles the work with the rest of the enterprise architecture for a purpose.

41. Governing Principles

Execution Architecture is the architecture of Instructions.

Execution Architecture defines the enterprise's instructions for how work is performed.

Execution Architecture is modular, collapsible and composable.

Process Module identity is established by its Canonical Invariant Contract.

Every Process Module recursively decomposes into Canonical Invariant Contract, Approved Variants, Tasks and Instructions as required.

If a proposed Variant changes the core Trigger, Bounded Purpose, Output, Completion Meaning or Essential Evidence, it is a different Process Module or module proposition.

A difference elsewhere in AEA does not automatically create an Execution Variant.

Source condition !’ Execution impact !’ Approved Variant where required.

Completion is a condition, not simply the end of a list.

Execution defines what evidence the work requires. Intelligence enables that evidence to be understood. Reality captures what actually exists.

Operational RACI belongs to Execution.

Operational RACI "` Workforce Authority Structure.

Operational Metrics belong to Execution. Actual metric values belong to Reality.

Execution defines the measure !’ Intelligence may establish the derivation !’ Reality captures the actual value.

Capability "` Process.

Relationship obligation "` Instruction.

Environment enablement "` Instruction.

Role "` Process.

Trust condition "` Instruction.

Audience Segment "` Execution Variant.

Lessons learned must inform governed change rather than uncontrolled process drift.

BOAS, SOP and User Journey are cross-architecture renders, not native Execution objects.

Intelligence enables assembled understanding. AEA in Action performs cross-architecture composition.

42. AI Protocol

AI may assist Execution Architecture by:

· identifying candidate Process Module boundaries;
· testing Canonical Invariant Contracts;
· identifying potential Variants;
· determining whether a proposed Variant changes invariant identity;
· decomposing Process Modules into Tasks and Instructions;
· identifying exceptions;
· identifying Operational Policy implications;
· extracting participation requirements;
· testing Operational RACI;
· identifying Operational Metrics;
· comparing intended and actual execution where Reality is available;
· identifying repeated exceptions or recurring variants;
· identifying candidate learning loops;
· and detecting where another architecture's proposition has been incorrectly absorbed into Execution.

AI must not:

· create a new Process Module solely because tooling changes;
· treat every contextual difference as a Variant;
· preserve a Process Module label where its invariant identity has materially changed;
· infer completion merely because all listed steps were attempted;
· invent evidence that does not exist;
· treat a contractual commitment as an Instruction without establishing the work response;
· treat Operational RACI as the complete Authority Structure;
· infer enterprise Authority from an Accountable RACI designation;
· treat Participant identity as Process identity;
· treat an Audience Segment as an automatic Variant;
· treat a Trust difference as an automatic Variant;
· treat actual execution as intended execution;
· silently rewrite instructions from lessons learned without governed change;
· treat BOAS, SOP or User Journey as native Execution objects;
· assign BOAS or SOP ownership to Intelligence;
· or expose proprietary AEA in Action composition methods as Published Standard doctrine.

AI may assist the design and interpretation of instructions. It must not manufacture process identity, completion, authority or controlled change.

43. Conformance

A conformant application of Execution Architecture must preserve, where applicable:

1. Process Module; 2. Canonical Invariant Contract; 3. Trigger; 4. Inputs; 5. Bounded Purpose; 6. Output; 7. Completion Meaning; 8. Evidence; 9. Exceptions; 10. Approved Variants; 11. Tasks where required; 12. Instructions; 13. Operational Policies where required; 14. Operational RACI where required; 15. Operational Metrics where required; 16. distinction between Process Module and Variant; 17. distinction between Capability and Process; 18. distinction between Operational RACI and Workforce Authority Structure; 19. distinction between metric definition and measured actuality; 20. source ownership of Variant-causing conditions; 21. Requirements of Understanding originating in Execution where applicable; 22. distinction between intended and actual execution; 23. and separation between native Execution objects and AEA in Action renders.

44. Non-Conformance

An application is non-conformant where it:

- treats Capability as Process;
- treats Role as Process;
- treats system or tool as Process;
- creates Process Modules without a coherent invariant identity;
- labels materially different work as a Variant after changing core Trigger, Purpose, Output, Completion Meaning or Essential Evidence;
- creates a Variant merely because Environment, Workforce, Ecosystem, Trust or Expression context differs;
- treats the end of a step list as proof of completion;
- treats evidence requirement and actual evidence as the same proposition;
- treats Operational RACI as Workforce Authority Structure;
- treats Operational Metric definition as the actual metric value;
- treats a Relationship commitment as an executable instruction;
- lets a named Participant determine Process identity;
- silently changes governed instructions because runtime learning suggests an improvement;
- treats a BOAS as a native Execution object;
- treats an SOP as a native Execution object;
- treats a User Journey as a native Execution object;
- assigns BOAS/SOP ownership to Intelligence;
- or exposes cross-architecture composition methods as though they were Execution doctrine.

45. Published Gaps

GAPS — None currently identified that prevent publication of this Standard.

Detailed Process Module fields, Variant schemas, Task and Instruction formats, Operational Policy specifications, RACI schemas, Operational Metric schemas, composition rules, native schematic construction protocols and purpose-specific BOAS/SOP/User Journey methods belong in the downstream Execution Architecture package or AEA in Action as appropriate.

Their absence from this Published Standard does not constitute an architecture doctrine gap.

46. Publication Position

Execution Architecture provides AEA with its architecture of Instructions.

Its core structure is:

Process Module !’ Canonical Invariant Contract !’ Approved Variant(s) !’ Task(s) !’ Instruction(s)

This structure enables the enterprise to define reusable work without either forcing every context into one rigid process or creating a different process for every contextual difference.

The Canonical Invariant Contract preserves identity.

Approved Variants preserve legitimate contextual execution differences.

Tasks and Instructions provide executable detail.

Operational Policies govern application.

Operational RACI defines participation requirements.

Operational Metrics define how intended execution is measured.

The architecture also preserves its boundaries:

Capability "` Process

Relationship commitment "` Instruction

Environment enablement "` Instruction

Role "` Process

Operational RACI "` Authority Structure

Trust condition "` Instruction

Audience Segment "` Execution Variant

intended execution "` actual execution

and:

BOAS / SOP / User Journey "` native Execution objects.

Execution contributes substantially to those operational renders, but does not own the whole enterprise mechanism they represent.

The governing application boundary is therefore:

Execution defines the work. Intelligence enables the understanding required to combine it with the wider Architecture Set. AEA in Action performs the cross-architecture composition.

And the governing architectural proposition remains:

Execution Architecture defines the enterprise's instructions for how work is performed.