Getting The Words Right

John Cutler

Contents

Introduction **1**
Why This Guide 1
Who Is This For 2
In This Guide 2

The Unfolding **3**

Endurant vs. Perdurant **4**
Other Names You May Hear 4
Diagram 5
Operational Difference 5
In Practical Modeling Terms 6
Short Rule of Thumb 6
Example Pair 7
Events, States, and Transitions 7
Ontology Mapping 8
Why This Matters Operationally 9

From Clear Flows to Complex Systems **10**
A Clear, Linear Case 10
A Common Product Flow 11
When Time Moves Forward 11
Modeling Implications 13
A Simple Contrast 14
Why This Matters 14

Static vs. Dynamic Optimization **16**
The Core Distinction 16
Other Names You May Hear 18
Operational Difference 18

Where Teams Get Into Trouble 20
Back to the Framework 20
Why This Matters 21

RAG Status, Endurants, and Perdurants **22**
In Manufacturing 22
In Knowledge Work 23
Where It Goes Off Track 24
Summary Containers 24
Why It Breaks Differently 25
The Core Insight 26

Containers vs. Anchors **27**
Containers 27
Anchors 27
Across Traditions 28
Why It Matters 29
Back to RAG 29
A Practical Contrast 30
A Simple Test 30
The Shift 31
Rule of Thumb 32
The Deeper Point 32

The Limits of Cascades **33**
Why Cascades Are Appealing 33
Operational Difference 34
What Cascades Hide 34
What To Model Instead 35
Why This Matters 37

Métis vs. Legibility **39**
Legibility 39
Métis 40
Across Traditions 40
The Mismatch 41
Compensation 42
A Practical Contrast 42
A Simple Test 43

Case Study: Event Storming Perdurants **44**
The Exercise 44

What the Wall Revealed 45
An Anonymized Slice 45
Containers, Anchors, and Verbs 47
The Left of the Ticket 48
Blast Radius 48
Promotion vs. Reality 48
Why Event Storming Helps 49
Beyond Big Picture Event Storming 49
The Core Insight 50

Theoretical SDLC vs. Real SDLC(s) 52
Why Theoretical SDLCs Appeal 52
Why Real SDLCs Diverge 52
A Practical Contrast 53
SDLC as Container 54
What To Standardize 54
The Core Insight 54

Transformation Journey as Ontology Shifts 56
What Is Shifting 56
Act-by-Act Pattern 57
Act 1: Work as Reality 58
Act 2 and 3: Goals Reorder the Model 58
Act 4: Contest Over Anchors 58
Act 5: Convergence 59
Containers, Anchors, and Overburdened Concepts 60
Why It Is Not Linear 60
The Core Insight 60

Coupling, Legibility, and Métis 62
Command Towers 62
Locked Grid 64
Federated Islands 65
Hands-Off Gridlock 65
The Cycle 66
Cross-Quadrant Insight 66
Endurants and Perdurants 68
Abstraction Quality 68
Final Synthesis 68

Factors Shaping Legibility and Métis 70
At a Glance 70

Scale of the Local 71
Endurant Fidelity 71
Perdurant Strategy 72
Mode of Métis Integration 72
Coupling Reality vs. Coupling Abstraction 73
Constraint Strategy 73
The Core Insight 74

Will It Scale? **75**
The Scaling Assumption 75
Where It Starts to Break 76
The Containment Problem 76
Some Things Are More Fractal 77
Organism, Not Bigger Cell 77
A Practical Test 78
The Core Insight 78

Context Is Not Just Transmitted **80**
Shannon vs. 4Es 80
Context Through Interaction 81
Why This Matters in Complex Environments 82
How the Approach Varies by Setting 82
Reprise 84
The Core Insight 85

AI Opportunities (and Caveats) **86**
At a Glance 86
Translation Layers 87
Messy Work and Exceptions 88
Keeping Artifacts Alive 88
Multiple Lenses, One System 88
Narratives, Relationships, and “My Dots” 89
Official, Real, Ideal 89
The Series Lens 90
The Core Insight 90

Twelve Practical Moves **92**
The Key Move 92
Eleven More Moves 95

Glossary **97**
How to Use This Glossary 97

At a Glance 97
Core Ontology Terms 99
Representation and Sensemaking 102
System Conditions and Design 104
Operational Frames and Practices 107

Introduction

Why This Guide

How many times have you heard an executive team complain that they have no visibility into what is happening? They are surrounded by slides, dashboards, RAG statuses, and data, but still do not really know what is going on.

How many times have you heard a team complain that they are overwhelmed, that they have no sense of the real priorities, and that they are in constant swivel-chair mode "reporting" on things without much confidence that the reporting matters, or reaches the right ears?

How often have you heard people talk about the workarounds they need just to get any work done? And how they dread the moment a leader assumes context that is not really there, because they saw a status somewhere that was meant to tell the truth, but got flattened, filtered, or whitewashed on the way up?

How often have you heard people complain that there were no set definitions for things, but then when someone finally did try to set those definitions, it took the life and variety out of the thing itself? What was once variable, contextual, and real became an oversimplification to placate someone who needed a neat dashboard.

The inspiration for this short guide came from a simple observation: operations, especially in complex domains, involves information architecture and ontology creation. But it is very different from what you might do if you were doing operations management in a factory or manufacturing setting. The concepts are slippier. The flows and feedback loops are more variable. What you call things is extremely important, but also much more of an art than in more deterministic settings.

Words alone will rarely sink the ship. But they absolutely can contribute

to headaches, misalignment, wasted motion, false confidence, and harmful flattening.

Who Is This For

This guide is for people working in and around product development operations, especially where the work is hard to name cleanly and harder to represent well.

It is for leaders and operators who have to decide what to call things, how to structure status, how to design planning and reporting views, and how to make complex work more legible without flattening it beyond usefulness.

That includes people in product operations, technology operations, transformation work, portfolio or planning functions, architecture, delivery leadership, platform leadership, and adjacent roles responsible for operating models, governance, coordination, or shared language.

In This Guide

In this guide, we will explore how people name work, how those names drift as they move across a system, how containers get mistaken for reality, how summaries lose context, and why so many operating problems are really problems of representation, interpretation, and scale.

The Unfolding

Summary: Product development is not best understood as a collection of fixed artifacts moving through fixed stages. It is better understood as something unfolding over time.

We tend to model software product development as a collection of things, passed across a fixed set of stages: goals, initiatives, epics, tickets.

But the work does not behave that way. It unfolds.

Discovery overlaps with delivery. Feedback reshapes priorities. What we are building, and why, changes as we learn.

The process does not just move forward. It changes as it progresses.

Even the context needed to understand the work is not fixed. It is produced through interaction, through the back-and-forth of people working through the situation together.

This requires a different orientation. Instead of organizing around static artifacts, we focus on what is unfolding over time. Instead of fixed stages, we look at loops, interactions, and shifts in state.

The goal is not to remove structure, but to use it in a way that stays closer to reality. To support learning, not mask it. To make work more legible without flattening it. To build representations that evolve with the system they describe.

Endurant vs. Perdurant

Summary: Use `endurant` for the thing that exists at a time, and `perdurant` for the process, event, or history that unfolds over time.

In ontology terms, an endurant is an entity that is wholly present whenever it exists. A perdurant is an entity that unfolds through time and therefore has temporal parts or phases rather than being wholly present at any one instant.

(Note: If you are French, or speak another Romance language where ***`endurant`*** *and* ***`perdurant`*** *already sound like they ought to mean something, this may start to feel almost too semantic. Fair enough. But this is not just wordplay. Some serious ontology people really did adopt these as technical terms. Here, they name a specific distinction between entities that are wholly present at a time and entities that unfold through time.)*

Other Names You May Hear

Different modeling traditions use different words for roughly similar ideas. The mapping is not exact, but it helps orient readers:

Ontology term	Roughly similar names
Endurant	`continuant` in BFO, or everyday modeling terms like `thing`, `entity`, `object`, `asset`, or `resource`
Perdurant	`occurrent` in BFO, or terms like `process`, `activity`, `occurrence`, `workflow`, or `episode`
Transition event	`domain event`, `state change event`, or `status change`

Ontology term	Roughly similar names
Event history	`event log`, `audit trail`, or `event stream`
State	`status`, `condition`, or `mode`

If you come from EventStorming or event-driven design, the easiest mental model is:

- the enduring business thing is the endurant
- the process unfolding over time is the perdurant
- the orange sticky-style business event is usually an event or transition event

Diagram

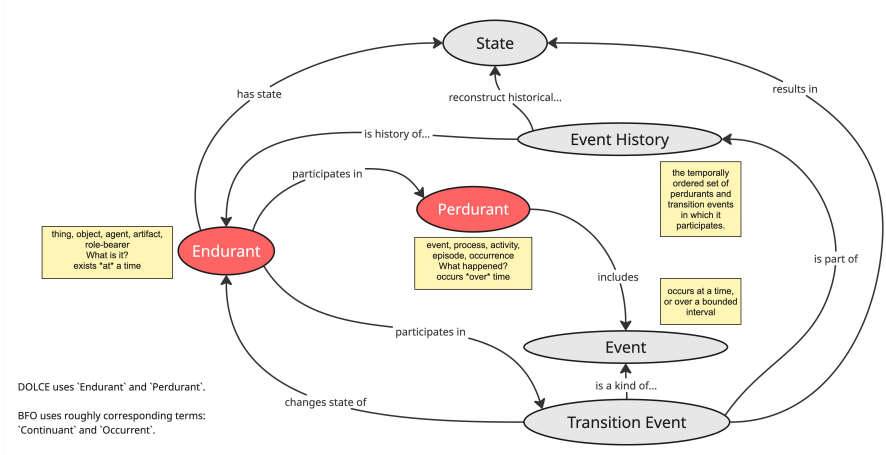

Figure 1: Endurant vs Perdurant diagram

Operational Difference

Use this test when modeling:

- If you can take a time-slice or snapshot and still say “the whole thing is here now,” model it as an endurant.
- If the thing only exists by happening over an interval and can be divided into before/during/after, stages, or episodes, model it as a perdurant.

In Practical Modeling Terms

Endurant

An endurant is usually a thing, object, agent, artifact, or role-bearer. It is the kind of entity you can point to at a given moment, even if some of its properties change over time.

- It persists through time while remaining the same entity.
- It may change properties over time.
- It participates in events and processes, but is not itself the event or process.
- It is typically modeled as something you can identify at a given moment.

Examples:

- a person
- a forklift
- a warehouse
- a contract document
- a customer account

Perdurant

A perdurant is usually an event, process, activity, episode, or occurrence. It is the kind of entity that only makes sense as something unfolding through time.

- It happens over time.
- It has temporal extent: a start, duration, and often an end.
- It can often be decomposed into phases, steps, or sub-events.
- Other entities participate in it.

Examples:

- a delivery
- a maintenance visit
- a hiring process
- a customer support call
- a system outage

Short Rule of Thumb

- Endurant: “What is it?”

- Perdurant: "What happens?"

Or:

- Endurant: exists at a time.
- Perdurant: occurs over time.

Example Pair

- `Technician` = endurant
- `Repair job` = perdurant

The technician is wholly present at each time at which the technician exists. The repair job is not wholly present at a single instant; it unfolds across diagnosis, parts replacement, testing, and completion.

Events, States, and Transitions

It is useful to distinguish the enduring thing from the events that affect it, because many modeling mistakes come from treating the entity and its unfolding history as if they were the same kind of thing.

- The endurant is the entity that has properties or states.
- The perdurant is the larger activity, process, or history unfolding over time.
- An event is a bounded occurrence within that history.
- A transition event is an event that changes the state of an endurant.

Manufacturing Example

- `CNC machine` = endurant
- `Production run` = perdurant
- `Machine status change` = event, more specifically a transition event
- `Idle`, `Running`, `Stopped`, `Maintenance` = states or status values of the machine

So the machine exists at a time, the production run unfolds over time, and the status change occurs at a specific time or over a very short interval. That distinction is what lets the model separate the thing being tracked from the change that happened to it.

Practical Modeling Rule

When something "changes at a certain time", do not treat the change itself as a property. Treat it as an event that results in a new property value, so the model can preserve both the current state and the transition that produced it.

For example:

- before the event, the machine has status `Idle`
- the `MachineStatusChangeEvent` occurs
- after the event, the machine has status `Running`

This lets you represent:

- when the change became effective
- what the old value was
- what the new value is
- who made the change
- why the change happened

Useful Terms

Depending on how formal or business-oriented you want to be, the "actual thing that happens" can be named in several different ways:

- `event`
- `sub-event`
- `occurrence`
- `state change event`
- `transition event`

In most operational models, `state change event` or `transition event` is the clearest term for a point-like change such as a status update, because those labels emphasize that something happened and that it changed the state of an enduring thing.

Ontology Mapping

In common foundational ontologies:

- DOLCE uses `Endurant` and `Perdurant`.
- BFO uses roughly corresponding terms: `Continuant` and `Occurrent`.

Why This Matters Operationally

If you confuse the two, your model tends to blur:

- stable entities vs. time-bound activities
- participants vs. the events they participate in
- object attributes vs. process states

That usually leads to poor queries, unclear event histories, and hard-to-maintain schemas, because the model stops distinguishing stable entities from the events and processes they participate in.

Try This Now:

- Write down one thing in your organization that passes the endurant test: you can point to it at a moment in time and still say "the whole thing is here."
- Then write down one thing your organization talks about as if it were a thing, but that really makes more sense as something unfolding over time.
- Ask: what gets distorted when we treat the second one like the first?

From Clear Flows to Complex Systems

Summary: In clear systems, endurants and perdurants map cleanly; in complex systems, evolving entities, overlapping processes, and state changes become much harder to model.

One way to build intuition for **endurants** and **perdurants** is to start with a system where the distinction is obvious, and then move to one where it begins to break down.

A Clear, Linear Case

Consider a simple manufacturing flow:

```
Raw materials -> Assembly -> Quality check -> Packaging ->
Shipping
```

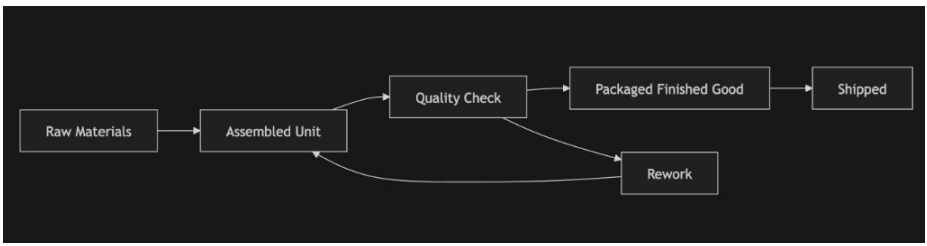

Figure 2: Manufacturing flow with rework path

This is a system where the structure is easy to reason about. The endurants are concrete and stable: parts, machines, finished goods. The perdurants are well-defined: assembly, inspection, packaging. The flow is largely linear, with work moving from one stage to the next in a predictable way.

Time and state exist, but they are not the primary challenge. In practice, you can often treat them as background conditions rather than as the central modeling problem.

- a unit is either assembled or not
- a machine is either running or idle
- a product is either shipped or not

You can model this system effectively without deeply modeling time, because a simple progression through stages is often sufficient. This is what a clear system looks like.

A Common Product Flow

Now consider a typical product development environment:

```
Insights -> Opportunities -> Options -> Bets -> Work ->
Releases -> Impacts -> Signals -> Adjustments
```

At first glance, this also looks like a flow, but it behaves very differently. The endurants are more abstract and fluid: things like opportunities, bets, initiatives, and goals are not physical objects. They evolve, get reinterpreted, split, merged, and sometimes disappear.

The perdurants are also overlapping and non-linear. Discovery, shaping, delivery, and learning are not clean stages, because they happen simultaneously and influence each other continuously. The structure is not truly linear.

There are multiple interacting loops:

- discovery loops
- delivery loops
- feedback loops

These loops intersect and interfere with each other.

Here is a more realistic picture of that kind of product flow:

Here is a real-world mapping of a team's product development flow. Notice all the loops, and the non-linear, multi-tiered nature of the system.

When Time Moves Forward

The key difference is not that manufacturing has time and state and product does not. The difference is where the modeling burden sits.

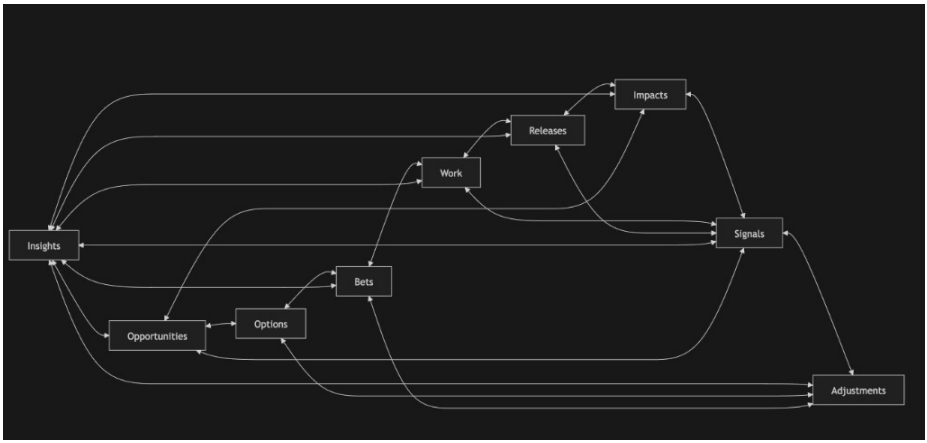

Figure 3: Product development flow as interacting loops

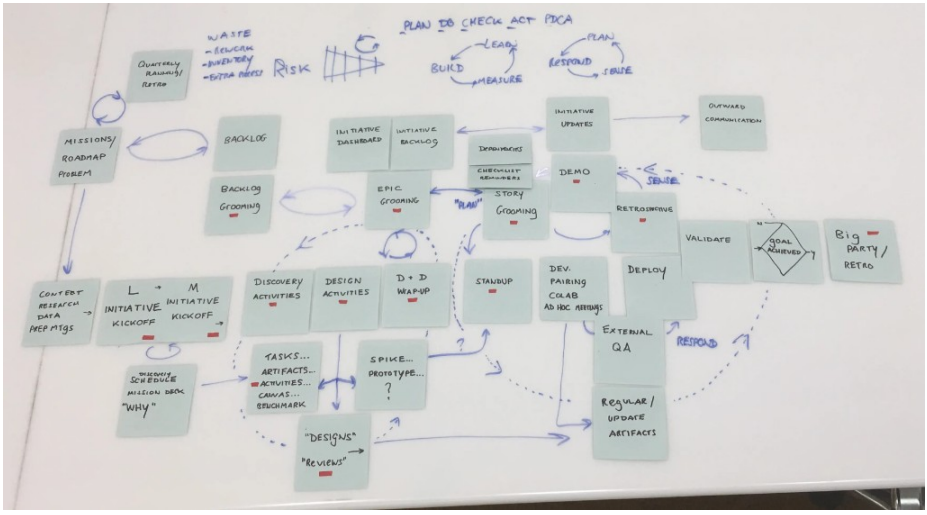

Figure 4: Whiteboard showing a team's product development flow

In clear systems:

- time and state are background concerns
- you can often treat them as implicit
- a simple stage-based model is enough

In complex systems:

- time and state become first-class concerns
- the meaning of things depends on when you look at them
- the same entity can be in multiple overlapping states

For example:

- an `opportunity` can be emerging, validated, deprioritized, or revived
- a `bet` can be tentative, active, paused, or abandoned
- `impact` may lag behind releases and be interpreted differently over time

You cannot fully understand the system by asking, "Where is this in the flow?"

You have to ask:

- what is the current state?
- how did it get here?
- what changed, and when?

Modeling Implications

This shift has direct implications for how you model. Endurants become more abstract, even though they are still "things." In complex settings they are conceptual and evolving:

- opportunity
- initiative
- metric
- goal

They persist, but their properties and meaning change over time.

Perdurants also become less cleanly bounded. They are no longer simple, linear processes:

- discovery work blends into delivery
- experiments overlap with releases

- feedback loops continuously reshape the system

The same perdurant may not have a clear start and end. As a result, events and transitions become critical, and you need to model:

- when something changed
- what the previous state was
- what the new state is
- why the change happened

Without this, the system becomes hard to reason about.

A Simple Contrast

A useful way to summarize the difference is this. In manufacturing, the question is, "Where is this unit in the process?" In product development, the question is, "What is the current state of this evolving entity, and how has it changed over time?"

Why This Matters

If you approach complex systems with a clear-flow mindset:

- you over-simplify the structure
- you hide important state changes
- you lose the history of how things evolved

If you recognize the shift:

- you treat endurants as evolving entities
- you treat perdurants as overlapping processes
- you model events and transitions explicitly
- you make time and state visible where they matter most

This is the point where endurants and perdurants stop being abstract ontology terms and become practical modeling tools. They become a way to decide what needs to be modeled explicitly, and what can remain implicit.

Try This Now:

- Write down one part of your organization where the stable thing and the unfolding process map cleanly.
- Then write down one part where the "thing" keeps changing shape and the process overlaps with other processes.

- Ask: where would you need explicit state, transitions, or history rather than a simple stage view?

Static vs. Dynamic Optimization

Summary: Treat product development as a dynamic optimization problem, not a static one. The difference is not academic. It changes how you define success, how you make decisions, and how you structure work.

The previous note showed why product systems are harder to model: the relevant things evolve, the processes overlap, and time and state matter more. This note adds a second lens. Once the landscape itself is moving, the problem is not just harder to represent. It is also a different kind of optimization problem.

The Core Distinction

A static optimization problem assumes:

- the objective is known
- the constraints are fixed
- the solution can be computed

A dynamic optimization problem assumes:

- the objective evolves
- constraints can be introduced, removed, or reshaped
- the solution is discovered over time

In product work, you are usually not solving for a fixed answer. You are navigating a moving landscape.

In more practical terms, static framing assumes you can define the goal, lock the constraints, make a plan, and then execute. Dynamic framing assumes you will learn what matters while doing the work, revise constraints to shape the search, and update decisions as new information

appears.

Aspect	More static optimization	More dynamic optimization
Definition	The objective and constraints are mostly known up front and stay stable long enough to solve against them.	The objective, constraints, or both may shift as the work unfolds and new information appears.
Objective function	Usually treated as fixed during the work.	Often revised as the team learns what matters or what is feasible.
Constraints	Mostly known in advance and expected to hold.	Can be introduced, relaxed, tightened, or reinterpreted over time.
Solution approach	More planning-heavy, analytic, and compute-oriented.	More adaptive, iterative, and learning-oriented.
Nature of the problem	Clearer boundaries and a more stable landscape.	Higher uncertainty, weaker initial definitions, and a changing landscape.
Decision making	Decisions are more front-loaded.	Decisions are revisited as signals, feedback, and conditions change.
Best fit	Stable environments with relatively dependable inputs.	Evolving environments where search and adjustment are part of the work.
Flexibility	Lower once the problem is framed correctly.	Higher because the framing itself may need to change.
Example	Tune a manufacturing line with known capacity and safety limits.	Explore product strategy or optimize a portfolio as conditions keep changing.

Other Names You May Hear

Different domains describe similar ideas in different ways:

Optimization framing	Roughly similar ideas
Static optimization	planning, prediction, upfront design, deterministic systems
Dynamic optimization	learning, adaptation, exploration, control systems
Objective function	success metric, goal, value function
Constraints	guardrails, policies, enabling constraints
Solution	plan, roadmap, implementation

These are not exact matches, but they help translate between theory and practice.

Operational Difference

Use this test:

- If you can define the objective, constraints, and solution before starting, and you have very high confidence they will all hold, you are in a more static problem.
- If any of those change meaningfully as you proceed, you are in a more dynamic problem.

Here is a rough gradient:

Case	Objective	Constraints	Solution shape
Very static: manufacturing throughput tuning	Increase throughput by a known amount	Mostly fixed: machine capacity, labor, safety limits	Choose and tune from a fairly well-understood solution space

Case	Objective	Constraints	Solution shape
More mixed: service operations improvement	Improve incident response and reduce customer pain	Some fixed, some redesignable: tooling, handoffs, ownership, alerting	Partly known upfront, partly discovered through investigation
Highly dynamic: new product or strategy work	The goal itself may shift as you learn	Useful constraints may need to be introduced gradually	Emerges through exploration, interpretation, and adjustment

Most consequential product work sits closer to the second and third cases than the first.

Example: Amazon's Weekly Business Review

A good example of dynamic optimization in practice is Amazon's WBR. It is often described as a metrics meeting, but it is more accurate to think of it as a system for continuously updating a model of the business.

- leaders review hundreds of metrics weekly

> "The metrics in a WBR are not static — you are expected to add or discard controllable input metrics as they stop working …"

As Cedric Chin argues in his piece on the Amazon WBR, it is best understood as a process control tool: a way to uncover and distribute the causal structure of a business so teams can act with better understanding.

From a modeling perspective:

- the business is treated as something unfolding over time
- metrics act as signals of that unfolding
- weekly review creates a learning loop

This is dynamic optimization in practice.

Where Teams Get Into Trouble

Most issues come from treating dynamic problems like static ones.

Common failure modes:

- locking the objective too early -> optimizing for the wrong thing
- treating constraints as fixed -> losing the ability to explore
- over-indexing on planning -> under-investing in learning
- chasing precision over direction -> local optimization, global confusion
- expecting one "right" solution -> ignoring viable alternatives

Back to the Framework

This connects directly back to the earlier notes in the series.

In clearer, more stable systems, the relevant endurants are easier to identify, the constraints hold more consistently, and the perdurants are easier to model as bounded flows. Static optimization makes more sense there because the landscape is not moving very much while you work.

In product work, the relevant endurants are often still being clarified, the constraints may be chosen or revised to shape the search, and the perdurant is the unfolding process of learning, deciding, shipping, interpreting signals, and adjusting. That is why the optimization problem is dynamic. The object of optimization is not fully settled when the work begins.

Seen this way:

- learning is part of the optimization process
- enabling constraints shape the search space
- flexible objectives acknowledge that success criteria evolve
- long-term thinking means optimizing across time, not just for the next local gain
- multiple viable solutions are normal, not a sign that the team has failed to choose the one right answer

This is not just a different set of tactics. It is a different class of problem.

As Things Become More Dynamic:

- **Focus on unfolding perdurants:** Model the things that happen over time, not just the things that exist. That usually means execution, discovery, learning cycles, and decision changes. These are

not secondary details. They are often the primary mechanism by which the system evolves.

- **Model enabling constraints explicitly:** Constraints are not just static rules. They are introduced, adjusted, and sometimes removed to shape how the system learns. Model when a constraint was introduced, what it was intended to do, how it changed over time, and what effect it had.
- **Focus less on labels, more on trajectories:** Whether something is called an `epic`, `initiative`, or `project` is often less important than how it unfolds. Shift from categorizing work toward understanding how it evolves, how it branches or converges, and how decisions shape its path over time.
- **Treat objectives as evolving, not fixed:** An objective is not always a stable target. Model how the objective was defined, how it changed, and what triggered the change. This preserves the context behind decisions and avoids rewriting history.
- **Preserve change as first-class:** When something changes, do not overwrite the past. Model the transition, the before and after, and the reason for the change. This allows the system to represent learning, not just current state.

Why This Matters

If you model a dynamic problem as static:

- your plans become brittle
- your metrics become misleading
- your teams optimize the wrong things

If you model it more accurately:

- you design for learning
- you adapt without thrashing
- you make better long-term decisions

Try This Now:

- Write down a current product initiative.
- Ask: which parts of this are actually fixed?
- Then ask: what are we pretending is fixed that is not?

RAG Status, Endurants, and Perdurants

Summary: RAG works when stable endurants map cleanly to visible perdurants, and breaks when containers and summary endurants hide the underlying dynamics.

One way to make this concrete is to look at a familiar tool: the RAG (`Red / Amber / Green`) status. At first glance, it seems universal, just a simple way to signal whether things are on track. But the meaning of that signal depends entirely on the relationship between the endurants, what you are tracking, and the perdurants, what is actually unfolding over time.

In Manufacturing

Consider a production line dashboard:

- `Line A: Green`
- `Line B: Amber`
- `Line C: Red`

In this setting:

- the endurants are stable and well-defined: machines, lines, batches, inventory
- the perdurants are structured and repeatable: production runs, assembly steps, quality checks
- the mapping between the two is tight

A production line has a known process, and a batch moves through known stages. Because of this, the RAG status is not just a color. It is usually backed by telemetry that carries context.

A red status likely corresponds to something legible:

- a machine failure
- a supply issue
- a quality threshold breach

And importantly, the system often makes it relatively clear what to do next. The signal is actionable without deep reinterpretation, because time and state are present but mostly implicit. The structure of the system does most of the work.

In Knowledge Work

Now consider a typical product or organizational dashboard:

- `Initiative A: Green`
- `Initiative B: Amber`
- `Initiative C: Red`

At first glance, this looks similar. But underneath:

- the endurants are more abstract and fluid: initiatives, bets, opportunities, strategic themes
- the perdurants are overlapping and evolving: discovery, delivery, alignment, learning, rework
- the mapping between the two is loose and shifting

What exactly is included in `Initiative A`? Has that changed? Is it still the same thing? In this context, the RAG status often carries very little embedded context.

A red status might mean:

- a dependency slipped
- a key assumption was invalidated
- scope changed
- alignment broke down
- new information emerged

But the color itself does not tell you:

- what changed
- when it changed
- why it changed
- what kind of response is required

The signal becomes a flag that something is off, without encoding the underlying situation.

Where It Goes Off Track

There are two common failure modes.

The Wrong Endurant

The thing being tracked is not a stable or coherent entity. A `project` or `initiative` may be:

- a bundle of loosely related work
- a shifting scope
- a container for multiple independent efforts

If the endurant is poorly defined, the RAG status becomes a flattened summary of unrelated dynamics. The underlying perdurants no longer map cleanly to it, and you are effectively asking, "What is the status of this thing?" when the thing itself is ambiguous.

The Missing Perdurant

Even if the endurant is reasonable, the unfolding process is not captured. There is no clear representation of:

- sequences of events
- transitions
- evolving states
- feedback loops

Without the perdurant, the RAG status becomes a snapshot without a story. Surprises become inevitable because:

- the path to the current state is invisible
- emerging risks are not legible

Summary Containers

There is an additional layer. Executives often ask for:

- the big picture
- a high-level view
- less detail

In the later terminology of this series, this is better understood as a **summary container**.

For example:

- portfolio status
- strategic theme status
- quarterly objective status

These are not the underlying things. They are containers that aggregate other endurants, each with their own perdurants. This creates a compounding effect.

A summary container sits on top of:

- multiple lower-level endurants
- multiple overlapping perdurants

The RAG status at this level compresses all of that into a single signal and often represents diffuse, distributed risk.

So when a leader sees `Theme X is Amber`, that status may reflect:

- small issues across many areas
- one critical issue hidden in the aggregate
- or simply uncertainty in interpretation

The risk is that the status appears precise, but the underlying reality is not.

Why It Breaks Differently

In manufacturing:

- endurants are physical and stable
- perdurants are structured and repeatable
- aggregations tend to preserve meaning, such as total output or defect rates

So even at higher levels:

- summaries remain grounded in consistent structures
- telemetry still carries actionable context

In knowledge work:

- endurants are constructed and evolving
- perdurants are non-linear and overlapping

- aggregations often erase the very context needed to interpret them

The Core Insight

A RAG status works when:

- the endurant is stable and well-defined
- the perdurant is structured and well-understood

It breaks down when:

- the endurant is ambiguous or shifting
- the perdurant is invisible or poorly represented

In those cases, the RAG status becomes a signal without sufficient context to support understanding or action. At best, it is a prompt to ask better questions. On its own, it is not an explanation.

Try This Now:

- Pick one status signal your organization uses regularly.
- Write down the thing that signal is supposed to describe. Does it feel like a real endurant, or is it actually a shifting bundle?
- Ask: what sequence, transition, or history would you need to see before the color would actually mean something?

Containers vs. Anchors

Summary: In complex systems, containers help organize intent, but anchors stay closer to reality and carry more reliable status.

Not all endurants are created equal. Some endurants act as containers, while others act as anchors. The distinction matters more as systems become complex.

Containers

A container is something we use to group and organize work. Common examples are `initiative`, `project`, `epic`, and `program`.

Containers are defined by inclusion. The implicit question is, "What's inside this?" They are useful because they create boundaries, help coordinate effort, and give a name to a bundle of intent.

In stable environments, containers can work well. Scope is relatively fixed, relationships are predictable, and the contents map reasonably well to reality.

In more complex environments, that starts to break down. Scope shifts, work moves in and out, dependencies cross boundaries, and meaning evolves over time. The container becomes a lagging, lossy summary of what is actually happening.

You can still assign a status to a container, but the status carries less and less context as the system becomes more fluid.

Anchors

An anchor is something that connects more directly to reality. Common examples are a `team`, a `production change`, a `metric`, or a `system` or

`service`.

Anchors are not defined by what they contain. They are defined by what they are and what happens to them over time.

You do not usually ask, "What's inside this?" Instead, you ask:

- what changed?
- what happened?
- what is the current state?
- how has it evolved over time?

Anchors persist through time, accumulate context, and remain meaningful even as work shifts around them. They become stable reference points in a changing system.

Across Traditions

This distinction overlaps with several adjacent traditions, even if they use different language.

Tradition	Container-like idea	Anchor-like idea	Typical failure mode
Information architecture	category, folder, navigation bucket	person, product, account, service	the navigation structure gets mistaken for the underlying reality
Taxonomy	classification bucket, class, grouping label	the entity being classified	the classification scheme starts to stand in for the thing itself
Knowledge graphs	collection node, grouping construct, context wrapper	durable entity node	grouping constructs get treated as if they had the same status as the underlying entities

Tradition	Container-like idea	Anchor-like idea	Typical failure mode
Portfolio management	initiative, program, epic, project	team, service, metric, production change	status gets attached to the wrapper instead of the more stable realities underneath
Systems thinking	abstraction layer, organizing frame	stable point of observation	the simplifying frame hides the actual dynamics of the system
Records or archives	collection, series, folder	record, item, artifact	the finding structure is treated as if it were the primary object of concern

Across all of these, the pattern is similar: containers help organize meaning, while anchors help keep us attached to reality.

Why It Matters

In simple systems, containers and anchors often align. A project maps cleanly to a set of activities, and a status on the container reflects something real about the underlying work.

In complex systems, containers drift away from the underlying work while anchors remain tied to observable reality. That creates a gap.

You might have:

- a green initiative
- while key metrics are degrading
- and recent production changes are causing issues

The container says one thing. The anchors say another.

Back to RAG

This explains a common failure mode with RAG status.

When you attach RAG status to a container, the signal compresses multiple, shifting realities. Context is lost, and interpretation is required.

When you attach signals to anchors, the signal is closer to reality. It connects more directly to action and carries more inherent meaning.

This is why:

- RAG works better in manufacturing
- struggles in knowledge work
- becomes especially fragile at the executive summary level

Because the further you move up, the more you rely on containers and the further you get from anchors.

A Practical Contrast

Example	Type	Why it matters
`Team`	Anchor	Persists, owns work, accumulates context
`Production change`	Anchor	Directly observable, tied to customer experience
`Metric`	Anchor	Reflects system behavior over time
`Initiative`	Container	Organizes intent, but drifts as reality changes

A Simple Test

If you are not sure whether something is acting more like a container or an anchor, use this side-by-side test:

Criterion	Anchor	Container
Is it directly observable in operations or outcomes?	Yes	No

Criterion	Anchor	Container
Can you ask "what changed?" and get a concrete answer?	Yes	No
Does it accumulate context over time in a stable way?	Yes	Weakly
Does a status signal on it usually imply a clear action?	Often	Rarely
Would it still make sense even if the current plan changed?	More likely	Less likely
Is it mainly defined by what it contains?	No	Yes
Can work move in and out without changing the label much?	No	Yes
Does its meaning drift as scope changes?	Less so	More so

If most of the answers line up with the `Anchor` column, treat it primarily as an anchor. If they line up with the `Container` column, treat it primarily as a container.

Some things will sit in the middle. The point is not perfect classification. It is to surface whether you are trying to explain reality with something that mostly organizes intent.

The Shift

As systems become more autonomous and interconnected, work becomes more fluid, processes become non-linear, and relationships become many-to-many.

Under those conditions, containers lose explanatory power and anchors become more important. This does not mean containers disappear, but it does change what they are good for.

Containers still help with:

- planning

- coordination
- communication

But they are no longer sufficient as:

- primary units of understanding
- reliable carriers of status
- proxies for reality

Rule of Thumb

If you want to understand what is going on, look at anchors. If you want to organize effort, use containers.

Confusing the two is where problems begin.

The Deeper Point

Most of the frustration you hear from leaders, "We don't have the big picture" or "We're surprised by what is happening," does not come from a lack of data.

It comes from relying on containers to explain a system that can only be understood through anchors and the perdurants connecting them.

Try This Now:

- Write down one container your organization uses often: for example an initiative, project, program, or epic.
- Then write down two anchors underneath it that are closer to reality: for example a team, service, release, metric, customer segment, or production change.
- Ask: if the container says "green" but the anchors do not, which one would you trust first?

The Limits of Cascades

Summary: Cascades and pyramids are useful as legible summaries, but they break down quickly as models of how product organizations actually work. They often turn into containers stacked on containers, hiding anchors, flattening perdurants, and making the system look cleaner than it is.

The previous notes established two important ideas. First, product systems are harder to model because the relevant entities evolve and the important processes overlap. Second, containers are not the same thing as anchors. This note takes the next step. Many of the cascades and pyramids organizations rely on are really containers stacked on containers. They create a legible representation, but they often drift away from the anchors and perdurants that would let people understand what is actually happening.

Why Cascades Are Appealing

Cascades are everywhere for a reason.

A cascade gives you:

- a visible hierarchy
- a sense of alignment
- a rough story about time
- a simple way to explain how work "ladders up"

That is useful, especially in large organizations that need a common representation.

In the language of this guide, a cascade is often a useful container. It gives people a portable, simplified view of how things relate.

The problem begins when those containers are stacked on top of each other and then mistaken for the underlying reality.

Operational Difference

Use this test:

- If the representation is being used as lightweight orientation, a shared shorthand, or a rough executive summary, a cascade can hold real value.
- If people are using it to explain how work actually moves, how outcomes materialize, how signals travel, or how decisions get revised, it will usually break down.

So the real question is not, "Is this a cascade?" It is, "What are we asking this representation to do?"

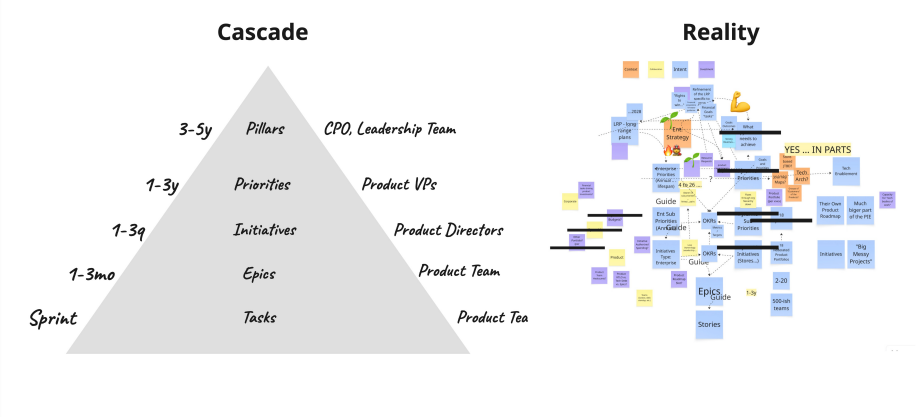

Figure 5: Cascade versus reality

A cascade offers a clean summary. The lived operating model is usually much more tangled, uneven, and multi-directional.

What Cascades Hide

Problem	Cascade suggests	What gets hidden
Direction	information mainly flows downward	feedback, learning, and signals moving upward from teams, customers, and operations
Time	each layer is a neat slice of the system	the perdurant: the history, overlap, iteration, and revision that make the work intelligible
Outcomes	work ladders up neatly across layers	outcome lag, uneven causal paths, and the fact that impact may materialize on very different timescales
Naming	each layer names a clean kind of thing	overloaded categories like `initiative`, `epic`, or `priority` that no longer match reality consistently
Participation	lower levels should become more specific	teams get cast as delivery recipients rather than participants in framing, interpretation, and strategy formation

That is why cascades often work better as explanatory shorthand than as serious operating models.

What To Model Instead

If you want something more faithful than a cascade, the move is not to create an even fancier pyramid. It is to make a few things more explicit.

A cascade often treats the named layers as if they were all stable endurants. But many of the most important things in product work are not stable in that way. `Initiatives`, `priorities`, `bets`, and even some

`goals` are often better understood as summaries of ongoing interpretation, coordination, and revision.

That is why cascades so often harden the wrong endurants and hide the perdurants that actually explain the system: discovery loops, decision cycles, learning loops, reframing, coordination work, and changing interpretations of impact.

Make explicit	Why it matters
anchors	so the model stays tied to things closer to observable reality
perdurants	so the important loops, histories, and changes do not disappear
transitions and state changes	so learning and reinterpretation stay visible
multiple lenses	so outcomes, work, finance, risk, and structure are not collapsed into one view
horizon and refresh cadence	so artifacts are understood by how far they look and how often they change

That usually gives you something less tidy than a cascade, but much more useful.

Case Study

A mid-sized SaaS company started with a classic cascade: a handful of company goals broken down into department goals, then into initiatives, epics, and tasks. It looked clean and aligned, but in practice it obscured how the business actually worked. High-impact work did not fit neatly into a single branch, goals drifted as conditions changed, and teams spent more time mapping work to the structure than understanding its impact. The cascade reinforced hierarchy and simplicity, but lost nuance around causality, timing, and how inputs actually drove outcomes.

The company shifted to modeling the business as a set of connected drivers. Goals were no longer tied to teams or levels, but to points of leverage in the system, with relationships between inputs and outputs made explicit. This resulted in

> more goals, not fewer, but those goals provided better context rather than more work. Instead of forcing alignment through structure, teams aligned through a shared understanding of how the system behaved and where they could have impact.
>
> The result was a shift from managing a plan to managing a system. Work was no longer categorized and cascaded, but evolved over time, influencing multiple goals and adapting as understanding improved. The company optimized for congruence over simplicity, treating product development not as a static plan to execute, but as a dynamic system to continuously learn and steer.

In series terms, the cascade had become a stack of containers mistaken for anchors. Some named layers were unstable endurants or even the wrong endurants, while the real perdurants, transitions, and outcome lag were getting compressed away. The better move was not less structure, but better structure: tighter anchors, more faithful endurants, explicit perdurants and state changes, and a form of legibility that stayed closer to reality instead of crowding out métis. In other words, they stopped treating a dynamic optimization problem as if it were a static cascade.

Why This Matters

If you treat a cascade as the real structure of the system:

- you hide upward feedback
- you hide the perdurants that explain how the system is actually moving
- you flatten multiple lenses into one
- you over-prescribe work at the edges
- you make flexible concepts look more stable than they really are
- you make small, high-impact work harder to represent honestly

If you recognize its limits:

- you make room for upward and sideways information flow
- you separate containers from anchors more carefully
- you preserve more of the unfolding process instead of only the summary
- you use multiple interacting views where one stack is too blunt
- you build a model that better matches how organizations actually learn and coordinate

The issue is not that cascades are useless. The issue is that they are too often mistaken for reality.

Try This Now:

- Take one common cascade or pyramid from your organization.
- Ask what it implies about direction: what seems to flow downward, and what important information is missing from the bottom up?
- Ask what lenses are absent: outcomes, work, finance, structure, risk, dependencies, or something else.
- Then ask: where are the real anchors, and which important perdurants or state changes are being compressed away?

Métis vs. Legibility

Summary: Legibility makes work visible from a distance, but métis is the situated knowledge that makes work actually function in complex settings.

There is a long-standing tension in how we represent work. One side values local, situated understanding. The other values clarity at a distance.

James C. Scott uses the term `métis` to describe the first: practical knowledge, built through experience, shaped by context, and constantly adapting. He contrasts it with efforts to make systems legible: simplified, standardized, and easy to read, measure, and control.

Legibility

Legibility has a clear appeal.

It allows:

- coordination across teams
- reporting and planning
- systems to be understood by people who are not directly involved

But it comes with a consistent move: it reduces complexity in order to make things visible.

That reduction is not neutral. It selects:

- what counts
- what gets named
- what gets ignored

Once something is made legible, it tends to be treated as if it is complete.

Scott's critique is not that legibility is bad. It is that legibility often replaces reality with a simplified version of reality, and then operates on

the simplification as if it were the real thing.

You see this in:

- standardized categories
- clean hierarchies
- formal plans
- system-of-record artifacts

Métis

Métis works differently.

It lives in:

- judgment calls
- adjustments in the moment
- shared understanding within a group
- things that are obvious locally but hard to explain globally

It does not try to eliminate ambiguity. It works with it.

In practice, much knowledge work sits much closer to métis than organizations often admit.

- priorities shift as new information comes in
- definitions evolve mid-stream
- the meaning of the work changes as it unfolds

Across Traditions

The same distinction appears in several adjacent traditions, even if they use different language.

Tradition	Legibility-like idea	Métis-like idea	Typical tension
Knowledge management	explicit knowledge, documented process	tacit knowledge, situated know-how	the documented version looks complete, but the real expertise remains local

Tradition	Legibility-like idea	Métis-like idea	Typical tension
Product and delivery	plan, roadmap, ticket hierarchy	judgment in discovery and delivery	the formal structure hides how the work actually changes
Systems thinking	simplified model, abstraction, control surface	local adaptation inside the system	the model coordinates action, but misses local dynamics
Anthropology or sociology	formal rule, official account	lived practice, situated behavior	the official story diverges from what people really do
Operations	standard operating procedure	operator adjustment and workaround	the procedure is legible, but the recovery work stays informal

Across these traditions, the recurring pattern is similar: legibility makes the system easier to see, while métis helps people cope with the reality the simplified view leaves out.

The Mismatch

The systems we use are overwhelmingly optimized for legibility. They assume:

- stable categories
- clear boundaries
- definitions that hold over time

That mismatch is where many familiar problems come from.

When legibility dominates too strongly:

- artifacts start to drift away from reality
- a ticket says one thing, but everyone "knows" what is really going on
- a status is technically correct but practically misleading

- a plan is clean, but the work behind it is anything but

At that point, the artifact is no longer a reliable representation. It becomes a simplified surface that requires interpretation.

Compensation

Teams respond in predictable ways. They do not abandon the system, but they stop relying on it alone.

They:

- restate things in multiple places
- add context in conversations
- maintain parallel documents
- re-explain what something "actually means"

This is not redundancy. It is compensation. They are rebuilding the context that legibility stripped away.

That is the part that is easy to miss. The issue is not just that legible systems are incomplete. It is that the more you rely on them as complete, the more you have to recreate reality somewhere else.

That "somewhere else" is where métis lives:

- in conversations
- in evolving documents
- in shared but unwritten understanding

A Practical Contrast

Legibility	Métis
makes things visible	makes things work
travels	adapts
simplifies for coordination	responds to local conditions
stabilizes categories	works with shifting situations

Both are necessary, but they are not interchangeable.

A Simple Test

The failure mode is not using legibility. The failure mode is treating legibility as if it were sufficient.

When that happens:

- systems look clean
- coordination appears smooth
- but the real work moves somewhere else

And once that gap opens, it tends to grow, because each additional layer of simplification creates more distance from the underlying situation.

A simple test:

- if you removed all conversations, would the artifacts still make sense?
- if you handed the system to someone new, would they understand what is really happening?

If the answer is no, you are relying on métis whether you acknowledge it or not.

Try This Now:

- Pick one artifact your organization relies on to make work legible: a dashboard, board, plan, status doc, or taxonomy.
- Then write down one place where the real work still depends on local judgment, side conversations, or tacit knowledge.
- Ask: what compensating behavior has grown around the artifact because it is not enough on its own?

Case Study: Event Storming Perdurants

Summary: A ticket often looks like a stable thing in a system, but Event Storming can reveal the much larger perdurant around it: the signals, conversations, decisions, splits, and changing interpretations that happened before, during, and after the row in the tool.

I did a fun exercise with a leadership team recently. We picked a couple teams, invited team members from each team, and together went methodically through a sample of the "tickets" they had been working on over the last quarter. We tried, to the best of our ability, to do a forensic analysis of sorts.

What made it so useful is that it quickly surfaced many of the themes in this series. The ticket looked like a stable endurant in the tool, but once people started talking, a much larger perdurant came into view.

The Exercise

Grounding: This section is mostly about `perdurants`: the sequence of decisions, interactions, and shifts through time that sit behind the more stable `endurants` people think they are discussing.

We asked questions like:

- How did this ticket come to be?
- What was the sequence of decisions that led to people paying attention to this "thing"?
- Who was involved? How were they involved, both implicitly and explicitly?
- Where did the "splits" happen, where one thing became multiple things?

- What language did people use to describe "it"?
- What happened afterwards?

We did a Big Picture Event Storming, inspired by the work of Alberto Brandolini, Paul Rayner, and others. The basic idea is to surface actors, decisions, where and how information changed shape, and the key moments that triggered those changes.

What the Wall Revealed

Grounding: This is where the note meets `métis` and `legibility`. The wall makes visible how much situated work exists before anything becomes clean enough to travel as a formal artifact.

I'll show a fully anonymized example below from the conversation, but while you're reading it imagine a bunch of stickies on the wall like `Signal raised in offsite`, `Follow-up to analytics`, `Cohort mismatch`, and `Onboarding flow suspected`. We ended up with `20+` sticky notes just for this one segment of the meeting.

That is part of the useful shock of the exercise. The tool may show one row, or a small cluster of rows, but the actual sequence behind that row is much more extensive, distributed, and socially constructed than most people assume. There was also something genuinely cathartic about making that hidden work visible.

This activity yielded a lot of work happening before something was ever placed in a ticketing system.

An Anonymized Slice

Grounding: Read this quote as the hidden history around a later artifact. In series terms, this is the `perdurant` that later gets compressed into a more legible `container`.

Here's an anonymized transcript from two minutes of the conversation centered around one collection of tickets:

> "If you remember, at the tail end of that meeting during the offsite, Sarah mentioned that activation rate for new enterprise accounts had dipped below 60%, and kind of half-directed that Mike should look into it. James was there as well. Nothing formally got logged, but later that afternoon Mike Slacked Priya in analytics like, 'hey, quick gut check,

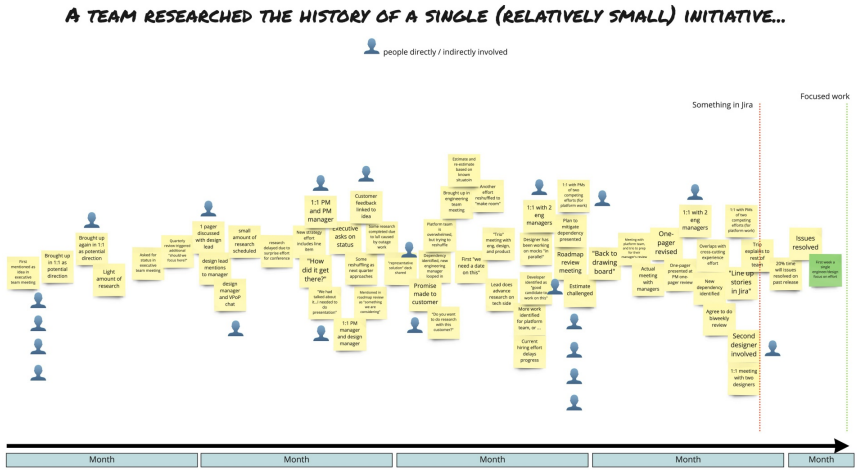

Figure 6: Event Storming wall showing the history of a small initiative before it appeared in Jira

are you seeing anything weird with activation on the new cohorts?' Priya pulled a quick cut that evening and noticed the drop seemed concentrated in accounts going through the new onboarding flow, but she was not totally sure because the definitions were not lining up cleanly. That turned into a back-and-forth the next morning, clarifying what counted as 'activation,' which cohort they were even talking about, and whether this was a real signal or just noise.

From there it lingered. It would come up briefly in staff meetings, 'are we still seeing that activation thing?' and then disappear again. There were a few more ad hoc cuts from analytics, a couple of quick product reviews, and some hallway conversations with engineering about the onboarding flow. Bits and pieces of work happened behind the scenes, but nothing fully pulled together. Over time it became a running topic, never quite top priority but never going away either.

After a couple of months, enough signal had accumulated and enough people had touched it that it finally got shaped into something more concrete. It was formalized into an OKR at the director level with Alex, framed around improving activation for new enterprise cohorts. That then got presented

> more cohesively to teams during the next offsite, where what had started as a passing comment turned into a shared focus with clearer ownership and intent."

We finally got to the "ticket," but by then everyone in the room could see that the ticket was only a very late and very partial representation of the work.

Containers, Anchors, and Verbs

Grounding: This section connects directly to the `containers vs. anchors` note. Terms like `initiative` and `objective` often behave like containers, while metrics, systems, teams, and observed changes act more like anchors.

When you do activities like this, a couple patterns become very clear. The words we use to describe things, `initiative`, `objective`, and so on, are mere pointers to containers of activity, decisions, conversations, focus, and so on. While we'd like to believe they are distinct things, the reality is that they overlap, merge, fork, replace, extend, and morph into each other.

Lots of teams stress out trying to define things, `bet`, `epic`, or whatever. "If we can all agree on what _____ means, then it will make it easier to collaborate!" That is true to an extent, but note what is often missing when these definitions emerge: actual activity, actual interactions, and actual information moving across time.

The definitions treat these concepts as nouns instead of verbs. `Initiatives` are a pointer to `initiative-ing`. `Objectives` are a pointer to `objective-ing`. Part of the reason people do this is that it is much easier to debate the definition of something than to wade into the murkiness of how doing the thing varies between teams and varies by context.

This is also where the earlier distinction between containers and anchors helps. The container terms drift and absorb meaning, while the more anchor-like realities, the metric movement, the onboarding flow, the team, the system under observation, stay closer to what is actually happening.

The Left of the Ticket

Grounding: This is another version of the `missing perdurant` problem from the RAG note. The row in the tool is real, but the path to that row is usually hidden, and that hidden path carries much of the meaning.

The rows in tools, the ticket in Jira, the OKR in a no-code tool, capture a tiny percentage of the overall context. In the example above, the "work" started with Sarah mentioning the metric movement at the offsite. Dozens, maybe hundreds, of hours were spent working on "the ticket," or tickets, before it became a row in a database somewhere.

Some of it was written down in docs. Some made it into comments. A slide here and there. But most of it is out there, floating in the ether. As a general rule, people are somehow surprised by how much stuff happens to the left, from a timeline standpoint, of the ticket, primarily because a lot of activity happens `1:1`, in silos, through informal channels, and out of sight of front-line teams.

Yes, some activity is associated with formal events like planning, check-ins, and reviews, but even that prep is largely unrecognized and, in many cases, underappreciated.

Blast Radius

Grounding: The blast radius is a practical example of `coupling` and `métis`. A small signal at one level can create large amounts of local interpretation, coordination, and invisible work elsewhere in the system.

With almost `100%` frequency, a senior executive will say something like, "Gosh, I didn't expect that one sentence to cause so much work. I was just asking about it! You could have told me to wait!"

That reaction matters because people are often not aware of the blast radius of even a seemingly innocuous request at any level above the team level. Event Storming helps make that propagation visible. It shows how attention spreads, how signals are amplified, and how loosely framed concerns can quietly create large amounts of follow-on work.

Promotion vs. Reality

Grounding: This section ties back to `legibility`. The promoted artifact becomes the official beginning of the story because it is the first

durable thing the system can point to, even when the real history started much earlier.

You also see a huge difference between things that are the units of promotion, visibility, funding, and so on, whether or not they feel real or useful, and all the other work that often goes unacknowledged and unobserved.

For example, in one case we discovered that the ticket could be traced back to a single proposal for headcount. In it, the director listed some options of things they could pursue. That somehow became part of the roadmap. There was a change on the team, and the new PM kept running with it. This was not part of any real planning ritual or discussion with stakeholders. It was a couple of lines written by someone trying to get some help.

That is another version of the legibility problem. The promoted artifact looks like the beginning of the story because it is the first thing the system can reliably point to. But it is usually not the beginning. It is just the first durable container that survived long enough to become legible.

Why Event Storming Helps

Grounding: Event Storming helps because it restores the relationship between `endurants` and `perdurants`. It gives people a way to inspect the evolving process around a thing instead of treating the later container as if it were the whole reality.

In this setting, Event Storming is useful because it reconstructs the perdurants around the artifact. It surfaces the sequence of signals, interpretations, conversations, decisions, splits, and reframings that the tool row compresses into a single object.

That makes the exercise valuable for operations people trying to model knowledge work. If you only model the containers, you miss the process through which meaning forms, changes, and spreads. If you only debate the definitions, you miss the actual interaction and information movement that gave rise to those definitions. Event Storming does not remove the ambiguity, but it makes the ambiguity more inspectable.

Beyond Big Picture Event Storming

Big Picture Event Storming is often the beginning, not the end.

One of the useful themes in Paul Rayner's EventStorming material, and in adjacent Event Storming practice, is that once you have a shared picture, you can zoom in. You can move from the broad historical wall into more focused views of the system: bounded contexts, business rules, commands, policies, aggregates, read models, and the actors or systems involved.

That matters for this series because at that point you are no longer just facilitating a conversation. You are starting to create an ontology for the domain. You are deciding:

- which events really matter
- what should count as stable enough to name
- what should be treated as an action, a state change, or a policy
- where one concept ends and another begins
- which abstractions are useful, and which ones are flattening too much

In more DDD-flavored terms, this is where Big Picture Event Storming can give way to design-level work. The wall helps people discover the major perdurants. The deeper work asks how those perdurants relate to commands, aggregates, business rules, and read models. In other words, the exercise starts to move from "what happened?" toward "what is the structure of this domain, and how should we model it?"

That deeper move is especially important in complex settings. You do not want the wall itself to become another frozen artifact. The value is not only in documenting the history. The value is in using that shared understanding to refine language, test boundaries, capture curated views, and decide what should remain local versus what should become a more durable part of the model.

So if you wanted to go deeper after a session like this, the next step would not just be "clean up the board." It would be to ask which parts of the picture deserve a more explicit ontology: which events recur, which commands trigger them, which policies connect them, which aggregates or entities they belong to, and which terms need tighter definitions because they are carrying too much ambiguity.

The Core Insight

Grounding: The larger series point is that knowledge work often fails when `containers` replace `anchors`, when `legibility` replaces `métis`, or when a visible `endurant` hides the more important `perdurant` around it.

Event Storming is not just a facilitation technique here. It is a way to make hidden perdurants visible. It shows that many of the things organizations treat as stable entities are really compressed summaries of longer, messier, more social histories.

That is why the exercise overlaps so strongly with the rest of this series. It exposes the gap between containers and anchors, between legibility and métis, and between the clean row in the system and the much larger reality that row only partially represents.

Try This Now:

- Pick one ticket, request, or initiative that looks straightforward in a tool.
- Write down three things that happened to the left of it: a signal, a conversation, a decision, a reinterpretation, or a split.
- Ask: how much of the real story is missing once the visible row becomes the official object?

Theoretical SDLC vs. Real SDLC(s)

Summary: A standard SDLC can be useful as a shared reference, but real delivery paths vary because team structure, uncertainty, timing, constraints, and decision authority vary.

Organizations often want a standard SDLC. The appeal is obvious: it creates a common language, a visible sequence, and a simpler way to coordinate. It gives people a legible model of how work should move.

That model can be useful, but only up to a point. In practice, there is rarely one real SDLC. There are many real SDLCs, each shaped by the conditions of the work.

Why Theoretical SDLCs Appeal

A theoretical SDLC helps with:

- planning
- governance
- reporting
- onboarding
- cross-team coordination

It gives the organization a stable container for talking about work. It says, in effect, "this is how work progresses here."

Why Real SDLCs Diverge

The problem is that the underlying work does not vary along just one axis. It varies across multiple dimensions at once.

Some of the biggest ones are:

- coordination load: one team working alone is very different from many tightly coupled teams
- time horizon: a few weeks is different from a multi-quarter or multi-year effort
- value timing: some work creates value early, while other work pays off much later
- uncertainty: sometimes the issue is execution, and sometimes the issue is whether the problem, approach, or outcome is even right
- de-risking cadence: some efforts can learn and pivot early, while others only reveal risk late
- constraints: some work has room to maneuver, while some sits inside tight limits
- urgency and value profile: not all work deserves the same level of protection, escalation, or investment
- decision authority and alignment: some work is team-owned, while other work depends on broader approval and convergence

Once those dimensions shift, the shape of the work shifts with them.

A Practical Contrast

Question	More theoretical SDLC view	More real SDLC view
What is the path?	one common sequence	multiple valid paths
What changes the path?	local variation around a standard	the nature of the work itself
How much discovery is needed?	assumed stage in the flow	varies by uncertainty, timing, and authority
How much coordination is needed?	often under-specified	depends heavily on team topology and coupling
When does risk come down?	expected to reduce in a predictable sequence	depends on what can actually be learned early
What does progress mean?	movement through stages	reduction in risk, creation of value, and alignment around next steps

SDLC as Container

This is where the earlier terms in the series help. A standard SDLC is often treated as if it were the real perdurant of delivery. But in practice, it is usually a container.

It groups many different real perdurants:

- discovery loops
- delivery loops
- coordination work
- decision cycles
- validation and learning

The container is useful because it creates a shared frame. But the real work unfolds differently depending on the system around it.

What To Standardize

This does not mean standards are useless. It means the organization has to be careful about what it standardizes.

It is often better to standardize:

- a small number of shared milestones
- explicit handoffs or interfaces
- key decision points
- common artifacts where consistency really matters
- signals that help others understand risk and progress

It is often worse to standardize:

- one rigid path for every kind of work
- one fixed discovery model
- one assumed coordination pattern
- one definition of progress for every initiative

The Core Insight

Theoretical SDLCs improve legibility. Real SDLCs reflect métis.

The mistake is not having a standard model. The mistake is treating that model as if it were sufficient. In knowledge work, the path depends on the work's actual conditions, and those conditions change the shape of the process itself.

So the practical question is not, "What is our SDLC?" It is, "What kind of work is this, and what kind of SDLC does it actually require?"

Try This Now:

- Pick one recent piece of work and write down the official path it was supposed to follow.
- Then write down the actual path it took, including any loops, waiting, rework, or exceptions.
- Ask: which parts need standardization, and which parts vary because the work itself varies?

Transformation Journey as Ontology Shifts

Summary: Transformation often looks like process change, but underneath it is a struggle over what the organization treats as real: work, goals, value models, teams, and the processes connecting them.

This note is an interpretation of TBM 402: The Real-World Journey to Value and Product-Centricity, read through the language of `endurants`, `perdurants`, `containers`, and `anchors`.

One useful way to read a transformation journey is not as a maturity model, but as a sequence of ontology shifts. The visible arguments are about frameworks, funding, planning, or org design. The deeper argument is about what counts as the primary unit of reality.

That is why these journeys feel so contested. The organization is not just changing process. It is renegotiating which endurants matter, which perdurants deserve attention, and which containers are being mistaken for anchors.

What Is Shifting

Across the journey, several things keep changing:

- what the organization treats as the main enduring thing
- what kind of progress it thinks it is tracking
- what gets rolled up for management
- what is treated as an explanation versus a summary

In one phase, `work` looks like the primary reality. In another, `goals` take over. Later, `value models`, `teams`, `platforms`, or customer-facing structures begin to matter more. Each shift changes the ontology of the

system.

Act-by-Act Pattern

Act	Dominant endurant	Dominant perdurant	Typical container	Core contention
Delivery predictability	work item, initiative, dependency	delivery flow, intake, throughput	project, initiative, work hierarchy	is moving work the same as creating value?
Early goals	work item plus goal	delivery plus reporting cycle	parallel work and goal hierarchies	are goals primary or just annotations on work?
Goals come first	objective, bet, opportunity	discovery, planning, delivery, learning	initiative as intervention	do goals actually ground action, or are they floating abstractions?
Emerging value models	goals, teams, journeys, capabilities, platforms	re-orgs, alignment, investment shifts, strategy debates	stacked models competing at once	what is the right anchor for investment and design?
Converging value models	value-bearing structures, products, platforms, teams	continuous adaptation around shared value logic	product and funding structures	how much alignment is enough before the model hardens again?

The important point is that the journey is not just about better tools. It is about changing what the organization believes the system is made of.

Act 1: Work as Reality

In the first act, work is the dominant endurant. The organization treats initiatives, projects, epics, and batches as the main things to manage. Progress means moving those things through stages.

The relevant perdurants are intake, delivery, dependency management, escalation, and throughput improvement. These are all real processes, but the model stays strongly container-oriented. The work item is treated as if it were the underlying reality rather than a packaging of intent.

That is why predictability gets conflated with value. The container moves, so the organization assumes value is being created.

Act 2 and 3: Goals Reorder the Model

The introduction of goals creates a second ontology on top of the first. Now the organization has at least two competing endurant-like structures:

- work hierarchies
- goal hierarchies

At first, goals are layered onto work. Later, goals come first and work becomes an intervention in service of them. This is a meaningful shift, but it also creates confusion.

The problem is that goals can easily become overloaded. They are asked to carry:

- intent
- strategy
- value
- measurement
- prioritization

That is too much for one concept. So goals become contested objects. They are more useful than raw work hierarchies, but they are not yet stable anchors.

Act 4: Contest Over Anchors

The fourth act is where the ontology becomes openly contested. Work, goals, value models, customer journeys, capabilities, platforms, and org structures all compete as organizing frames.

This is not just a communication problem. It is a disagreement about what the enduring entities of the system actually are. Are the important endurants:

- projects?
- objectives?
- products?
- platforms?
- customer journeys?
- stable value nodes?

At the same time, the perdurants multiply:

- strategy cycles
- planning cycles
- funding decisions
- re-orgs
- architecture refactoring
- discovery and delivery loops

This is why the stage feels chaotic. The organization is trying to run several ontologies at once.

Act 5: Convergence

In the fifth act, some of that contention settles. Value models begin to anchor the rest of the system. Teams, funding, goals, architecture, and planning start to line up around more stable value-bearing structures.

This does not eliminate change. It changes what is treated as stable enough to organize around. Goals stop floating on their own. Work stops being the sole unit of reality. More of the system attaches to anchors that are closer to how value is actually created.

At that point:

- goals attach to value structures
- work flows through those structures
- investment references them
- architecture reflects them

The ontology becomes more coherent, even if it never becomes final.

Containers, Anchors, and Overburdened Concepts

One of the strongest patterns in the piece is how concepts get overburdened over time.

The organization starts with one container, such as `work`. Then it introduces another, such as `goals`. Then it tries to make that new container do too much. Eventually another layer appears, such as `value models`, to carry what the earlier concept could not.

This produces a recurring pattern:

- a concept is introduced
- it becomes overloaded
- it is stretched to carry adjacent meanings
- it splits or is displaced
- a new concept emerges

That is why transformation language often feels unstable. The words are not just labels. They are being asked to carry competing models of reality.

Why It Is Not Linear

Seen this way, the journey is not linear because ontology shifts are not linear. Different parts of the organization can live in different models at the same time.

One group may still treat work as reality. Another may treat goals as primary. Another may already be organizing around value models or product structures. The organization is therefore not moving through one sequence so much as hosting multiple competing realities at once.

That is also why regression is common. A new market condition, new leadership idea, or technology shift can make the current ontology less useful and reopen the argument.

The Core Insight

Transformation is not just a change in workflow. It is a change in what the organization treats as real, what it treats as stable, and what it treats as explanatory.

That is why these journeys are full of contention. People are not merely debating process. They are debating whether `work`, `goals`, `value`, `teams`, `platforms`, or some other structure should be treated as the primary endurant of the system, and which perdurants actually explain progress over time.

Try This Now:

- Write down one word people in your organization argue about a lot: for example goal, product, platform, stream, or value.
- Then write down what that word seems to mean in two different parts of the organization.
- Ask: what different reality is each group trying to stabilize when they use the same word?

Coupling, Legibility, and Métis

Summary: Coupling changes how much coordination a system requires; legibility and métis are two different ways of carrying that burden, and each quadrant fails in a different way.

This quadrant model is useful because it shows that friction is not just about process quality. It is also about how much of the coordination burden is being carried by formal abstractions versus local judgment.

Read through the language of this series, the diagram is partly about control and autonomy, but more deeply about whether the system's `endurants`, `perdurants`, and abstractions match reality closely enough to support the work.

A coupling quadrant can be read as a map of where legibility dominates, where métis dominates, and where each one starts to fail.

Command Towers

This is the most legibility-first quadrant.

In this setting:

- legibility dominates
- métis is suppressed
- the endurants are large, stable, and imposed: teams, functions, projects
- the perdurants are standardized, top-down workflows

Work is shaped into clean, stable containers. Processes are predefined and expected to repeat. The system assumes nouns that behave and verbs that repeat.

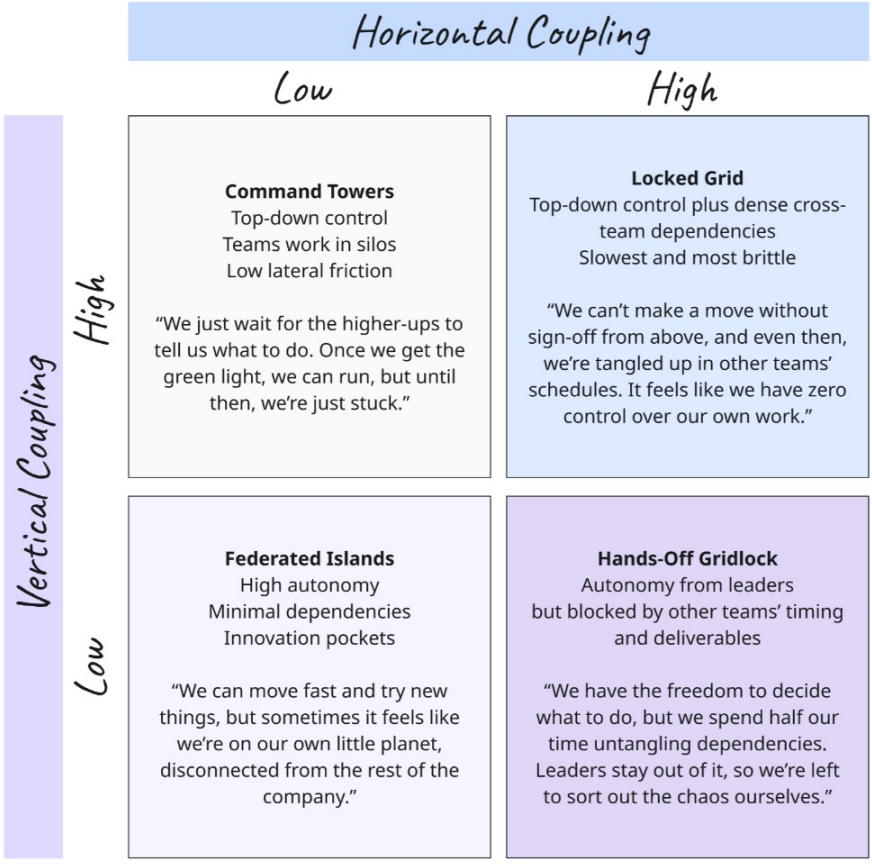

Figure 7: Coupling quadrants showing command towers, locked grid, federated islands, and hands-off gridlock

This works as long as:

- the abstractions are close enough to reality
- the work is not too dynamic

In knowledge work, the problem is that local context gets compressed out. Teams wait for decisions because the system must update first.

Friction type:

- waiting friction
- suppressed adaptation

Failure mode: the model becomes the bottleneck. The system is legible, but not responsive.

Locked Grid

This is the most structurally constrained quadrant.

In this setting:

- attempted legibility dominates under extreme coupling
- métis is overwhelmed
- the endurants are very large and highly entangled: cross-team initiatives, programs
- the perdurants are interwoven and non-linear in reality, but forced into linear stages

The "thing" spans many teams. The "flow" is deeply interdependent. Legibility is pushed hard to control the system.

But:

- the abstractions are often lossy
- the process cannot actually be linearized

So both begin to break down. Métis struggles too, because the local unit is now too large to reason about effectively.

Friction type:

- coordination friction
- translation friction

Failure mode: everything must be coordinated, but nothing is accurately represented. This is maximum brittleness: the system is both tightly coupled and poorly modeled.

Federated Islands

This is the most métis-friendly quadrant.

In this setting:

- métis dominates
- legibility is minimal
- the endurants are small, local, and often implicit
- the perdurants are emergent, adaptive, and team-specific

Teams operate with strong local context. Definitions evolve as needed. Work is shaped around real conditions rather than imposed categories.

Legibility still exists, but lightly:

- minimal shared structures
- weak global rollups

Friction type:

- low local friction
- occasional global misalignment

Failure mode: local systems work, but the broader organization lacks a strong shared picture.

Hands-Off Gridlock

This is the most unstable quadrant.

In this setting:

- neither legibility nor métis fully works
- effective legibility is missing
- métis is stretched across too many boundaries
- the endurants are cross-team but poorly defined
- the perdurants are highly interdependent and negotiated in real time

Work requires coordination across teams, but no strong legible system exists to support it. So teams rely on:

- constant conversation
- ad hoc negotiation
- re-interpretation of shared work

But:

- no shared abstraction stabilizes meaning
- no structure reduces the coordination burden

Friction type:

- continuous coordination friction
- high cognitive load

Failure mode: perpetual negotiation. The system depends on métis, but spreads it too thin to be effective.

The Cycle

The important thing about the quadrants is that organizations do not usually stay in one of them. The pattern is often cyclical.

One common sequence looks like this:

- `Locked Grid` becomes unbearable because coordination and abstraction both break down
- leaders respond by pulling toward `Command Towers` to reset control
- that reset reduces ambiguity, but eventually becomes too top-down and too slow
- pressure then pushes the system toward `Federated Islands` so teams can move again
- over time, dependencies accumulate across those islands
- the system drifts toward `Hands-Off Gridlock` and then back toward `Locked Grid`

The quadrants often behave less like destinations and more like a recurring movement between control, autonomy, and accumulating coordination burden.

What changes from one turn of the cycle to the next is not just the amount of control. It is also the quality of the abstractions the system is using.

If the abstractions improve, the cycle can loosen. If they remain poor, the organization keeps rediscovering the same friction in different forms.

Cross-Quadrant Insight

This diagram is really about how organizations distribute the burden between legibility and métis under different coupling conditions.

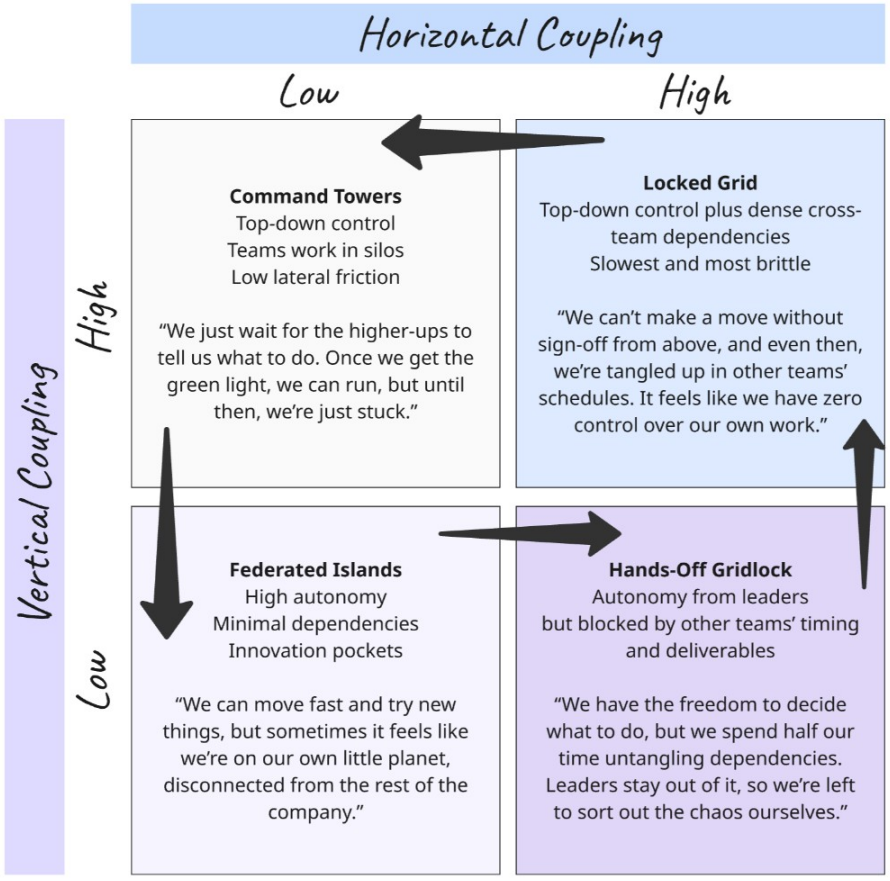

Figure 8: Coupling quadrants with arrows showing a recurring movement between states

Horizontal coupling increases the need for shared understanding. It pushes toward legibility, shared anchors, or a wider local scope for métis.

Vertical coupling increases imposed structure. It pushes toward standardized endurants and perdurants, but also risks over-compressing reality.

Endurants and Perdurants

Across the diagram:

- endurants become smaller and more local in low-coupling settings
- endurants become larger and more abstract in high-coupling settings
- perdurants stay more emergent and adaptive in low-coupling settings
- perdurants are more likely to be forced into standardized sequences in high-control settings

The key issue is not just the amount of coupling. It is whether the endurants and perdurants being used in the model actually match reality.

Abstraction Quality

This cuts across all four quadrants.

Good abstractions:

- reduce friction
- support both legibility and métis
- let local judgment and shared structure reinforce each other

Poor abstractions:

- increase translation work
- force teams to rebuild context elsewhere
- make both legibility and métis work harder than they should

That is why `Locked Grid` becomes unbearable and `Hands-Off Gridlock` becomes chaotic.

Final Synthesis

Coupling determines how much coordination is required. Legibility tries to make that coordination manageable. Métis fills the gaps where legibil-

ity falls short.

The tighter version is this: when coupling rises, the system must either improve its abstractions or rely more heavily on métis. If it does neither, friction explodes.

Try This Now:

- Pick one area of work that regularly crosses teams or functions.
- Ask which quadrant it feels closest to right now: Command Towers, Locked Grid, Federated Islands, or Hands-Off Gridlock.
- Then ask: is the coordination burden being carried mostly by good abstractions, or by people compensating in real time?

Factors Shaping Legibility and Métis

Summary: The balance between legibility and métis is shaped by a small number of underlying factors: how large the local is, what gets stabilized, how time is structured, how understanding is integrated, whether coupling is real, and where control lives.

One way to make the earlier notes more practical is to name the factors that keep changing the balance between legibility and métis. These factors help explain why one organization can operate with light structure while another collapses without it, and why the same practices work differently in different settings.

At a Glance

Factor	Core question	Common failure mode
Scale of the Local	How much reality must a team understand to act well?	pretending the local is smaller than it is
Endurant Fidelity	Are the stable containers close to reality?	lossy summaries and constant translation
Perdurant Strategy	How is time structured across kinds of work?	forcing all work into one cadence
Mode of Métis Integration	How does understanding actually move?	artifacts and rituals exist, but integration stops happening

Factor	Core question	Common failure mode
Coupling Reality vs. Coupling Abstraction	Are dependencies real or manufactured?	solving abstraction problems as if they were system problems
Constraint Strategy	Where does control actually live?	over-modeling and illusion of control

Scale of the Local

This factor asks: how large is the slice of reality a team must understand to do its job?

- small local: a team can reason end-to-end
- large local: useful understanding requires coordination across many teams and systems

In some settings, the `métis radius` is inherently large. A team cannot make a meaningful move without understanding platform constraints, infrastructure realities, upstream systems, downstream implications, and adjacent teams.

That is important because sometimes complexity is real, not a modeling failure.

Failure mode:

- pretending the local is small
- pushing autonomy when the métis radius is inherently large

Endurant Fidelity

This factor asks: how well do the stable containers reflect reality?

- high fidelity: teams, domains, funding units, or outcomes map reasonably well to the real system
- low fidelity: work is forced into projects, phases, or arbitrary buckets

This is about what the organization chooses to stabilize. If the wrong endurants become legible, the whole system pays a translation tax.

Example:

- ongoing or exploratory work forced into `projects`

- teams repeatedly re-describing their work to fit the container

Key insight: legibility is not neutral. You can make the wrong things legible.

Failure mode:

- lossy summaries
- constant translation and copy-paste work
- a frenetic operating environment

Perdurant Strategy

This factor asks: how does the organization structure change over time?

- mono-perdurant: one cadence or one process for everything
- plural perdurants: different rhythms for different kinds of work

Discovery, delivery, maintenance, migration, incident response, and platform work do not all unfold the same way. When the organization uses one cadence to govern all of them, it confuses uniformity with control.

Example:

- discovery, delivery, and maintenance all forced into the same sprint cadence

Key insight: uniformity is not the same thing as control.

Failure mode:

- heterogeneous work forced into one temporal model
- teams simulate compliance rather than operate effectively

Mode of Métis Integration

This factor asks: how does knowledge actually move and get integrated?

There are at least three broad modes:

- representational: documents, specs, models, and synthesis carry understanding
- relational: meetings, rituals, and working sessions carry understanding
- abandoned: artifacts exist but are not trusted, and meetings do not really integrate anything

Each mode can work if it is invested in. The problem is not whether a system is artifact-heavy or interaction-heavy. The problem is when the organization stops truly integrating understanding.

Failure mode in the abandoned mode:

- hidden dependencies
- rework and surprises
- chronic misalignment

Key insight: you either invest in integrating métis, or you slowly stop doing it.

Coupling Reality vs. Coupling Abstraction

This factor asks: are dependencies real, or introduced by the way the organization models work?

- real coupling: the system is genuinely interdependent
- artificial coupling: the dependencies are created by weak abstractions

Some coordination is necessary because the systems are tightly linked. Other coordination exists only because the organization put multiple teams inside the same project, reporting line, workflow, or planning container.

Examples:

- real: shared platform constraints or tightly linked systems
- artificial: teams forced to coordinate because they share a project or portfolio wrapper

Key insight: not all coordination is necessary. Some of it is manufactured.

Failure mode:

- solving abstraction problems as if they were system problems

Constraint Strategy

This factor asks: where does control actually live?

Two broad patterns show up:

- broad, shallow control: many rules, many structures, many attempts to make everything legible

- narrow, deep control: a small number of shared mechanisms, applied with real discipline

The second approach often works better in complex environments. It does not try to make everything legible. It chooses a few anchors and makes those reliable.

Example:

- strong discipline around a small set of operating anchors, metrics, or review mechanisms

Key insight: you do not need to make everything legible. You need to pick the right things.

Failure mode in the broad, shallow case:

- over-modeling
- fragile systems
- illusion of control

The Core Insight

These factors are useful because they keep the conversation from collapsing into slogans like "be more autonomous" or "standardize more." The real question is what kind of system you are dealing with, what kind of endurants and perdurants it contains, and what kind of legibility and métis it actually requires.

Try This Now:

- Pick one team or operating area and ask how large its local really is.
- Write down what the system has stabilized as the key endurant there, and whether that feels faithful to reality.
- Ask: where does control actually live, and how much translation work is needed to keep things moving?

Will It Scale?

Summary: Things do not scale the way people often imagine they scale. Some local models stretch. Some turn into containment devices. Some look scalable only because they add legibility while losing contact with reality.

There is a natural tendency, when thinking about scale, to imagine larger versions of the same constructs. Bigger efforts. Longer horizons. Still moving left to right through stages.

Sometimes that works. If the thing really behaves like coordinated execution, the model can stretch. Large efforts, many contributors, shared timelines. In those cases, it is still "work," just expanded.

But the further upstream you go, the less that assumption holds.

The Scaling Assumption

As scale increases, people often keep the same nouns and the same verbs.

- a team-level `epic` becomes a `Portfolio Epic`
- a local intake conversation becomes a portfolio `intake` stage
- a team workflow becomes a governance model

In that sense, scale is partly an `endurant` problem and partly a `perdurant` problem. People assume the same named things and the same flow structures remain valid as the resolution changes.

That move preserves `legibility`. It lets the organization say, "we already know what this is." But it hides the harder question: does the larger thing actually behave like the smaller thing?

Where It Starts to Break

The further upstream you go, the more you encounter things that do not behave like discrete units moving through a system.

This is where the earlier focus on `perdurants` matters. Some processes stay coherent as they expand. Others stop behaving like staged items and start behaving like:

- signals
- interpretation
- accumulation over time
- splits and recombination

Direction-setting, emerging understanding, and half-formed ideas do not move cleanly from one stage to another.

You see this almost immediately when doing Event Storming. As you move left, you leave the world of bounded items and enter something less structured. Instead of pieces advancing through steps, you see fragments of context, conversations, and decisions gradually taking shape until something solid enough emerges to act on.

That is where the abstraction starts to strain.

The Containment Problem

This is where `containers`, `anchors`, and `legibility` become useful.

At larger resolutions, organizations often add layers of containers because they are easier to summarize, even when those layers are not the most real thing in the system.

This shows up in language like `intake` at the program level. What is often a local, collaborative activity gets reframed as something centralized and transactional. Work is expected to arrive as discrete inputs, ready to be routed and assigned.

That can work in some contexts. But it assumes opportunities are formed before the people doing the work are involved. That is exactly where things start to break down. It is a good example of taking something inherently local and projecting it as if it scales cleanly.

The same thing happens with cascading OKRs.

- at the local level, a team may be working with something real enough to guide action
- as layers are added, the upper-level OKRs often become wrappers for wrappers
- each layer becomes more legible and less real

The system gains hierarchy, but loses fidelity.

Instead of flattening the problem to the relevant anchors, the organization adds more layers of things that are not especially real, simply because those layers are easier to name, route, and report.

Some Things Are More Fractal

Not everything fails under scale.

Some structures travel up and down because the pattern stays coherent even as the content changes. A coherent one-pager style can often scale surprisingly well, not because the content is the same, but because the structure is. The same shape can hold a team decision, a strategic choice, or a broader organizational proposal as long as each version is grounded in something real at its own level.

Input metrics can also scale more effectively than many planning containers. A good input metric can connect local action to broader system behavior without forcing everything into a containment hierarchy. It can move up and down if it continues to point to a meaningful signal rather than becoming a summary wrapper detached from operations.

That is the key distinction. The question is not whether something appears reusable. The question is whether it stays anchored as it moves across resolutions.

Organism, Not Bigger Cell

It is like assuming an organism behaves like a scaled-up cell.

- at the cellular level, processes are relatively contained and local
- at the level of an organism, you have signaling systems, competing priorities, feedback loops, and coordination across many subsystems

That is another way of describing `coupling` and the limits of local models. As scale changes, the coordination burden changes, and the larger system may have properties the smaller one simply does not.

The behavior is no longer just a bigger version of the same thing. The system has changed resolution, and with that change comes a different kind of coordination problem.

A Practical Test

If you think something scales, ask:

- is this the same kind of thing, or just a larger label?
- does the process still behave like one coherent perdurant?
- are we adding containers for legibility, or preserving anchors that stay close to reality?
- does this structure still help people act, or does it mainly help other people summarize?
- is the pattern truly reusable, or are we projecting a local success onto a different resolution?

The Core Insight

Things do not scale the way people often think they scale.

Scale failures often come from treating `legibility` as if it were fidelity. The model gets cleaner as it moves upward, but the reality it refers to gets thinner.

Some local models stretch because the underlying reality is still coherent. Some do not, because the larger resolution contains:

- more signaling
- more interpretation
- more coupling
- more contested meaning

than the smaller model can carry.

That is why scale so often becomes a containment problem. The organization keeps adding layers of things that are not especially real because they make the system easier to read. A better move is often to ask what can stay fractal, what should flatten to anchors, and where a different model is needed altogether.

Try This Now:

- Pick one model, ritual, or artifact that works well for a single team.
- Now imagine using the same thing across five or ten teams.

- Ask: which parts would still hold, and which parts would become thinner, more performative, or more misleading at that scale?

Context Is Not Just Transmitted

Summary: The Shannon-style view treats context as something that can be packaged and sent, while the 4E view treats understanding as something that emerges through interaction; that difference matters most in complex, evolving environments where endurants, perdurants, and meaning itself are still shifting.

One of the most important distinctions in this whole series is that context is not just something that is. In many settings, context is produced through interaction. That changes how we think about communication, coordination, and the role of AI.

It also connects directly to several earlier themes in the series:

- `legibility`: the attempt to make context portable, stable, and readable at a distance
- `métis`: the situated understanding that forms in action
- `endurants`: the things the organization treats as stable enough to reason about
- `perdurants`: the unfolding interactions, decisions, and adjustments through which understanding changes

The key shift is from treating context as portable overlap to treating context as something that emerges through interaction.

Shannon vs. 4Es

The Shannon-style model treats communication as a transmission problem.

- someone has a message

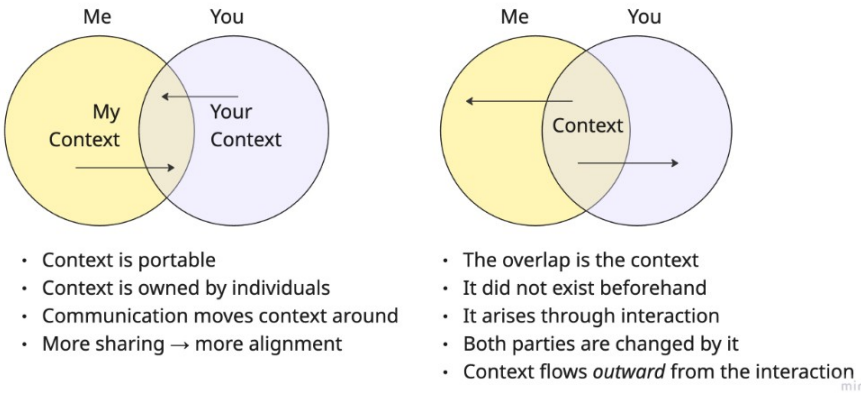

Figure 9: Two diagrams comparing portable context with interaction-generated context

- context helps decode it
- noise gets in the way

In that frame, better understanding comes from sending the right information with enough context attached. It is naturally attractive to legibility-first systems because it treats context as something that can be packaged, transported, and decoded.

The 4E view points somewhere else. Understanding is not just the reconstruction of a message. It is shaped by bodies, tools, environments, and interaction.

- `embodied`: contact with reality matters
- `embedded`: the environment shapes what can be seen and done
- `extended`: thinking is distributed across people and tools
- `enactive`: understanding emerges through interaction

The meeting, the working session, or the coordination ritual does not just transfer context. It generates it. That is much closer to the logic of métis, where understanding is formed in the situation rather than merely retrieved from a container.

Context Through Interaction

This matters because the portable-context model quietly assumes that people already have what they need, and only need to exchange it.

But in many real situations:

- the problem is still being framed
- the situation changes as people act
- interpretation shifts through disagreement and adjustment
- shared understanding does not exist beforehand

That is especially important in the language of this series. In complex settings, the endurants are evolving, the perdurants are unfolding, and the meaning of both changes as people engage with the work.

Context is therefore not just background. It is part of the process by which the system becomes legible to itself. In other words, the perdurant of interaction helps stabilize what the relevant endurants even are.

Why This Matters in Complex Environments

In complex environments, action and understanding co-evolve.

- people act
- the system changes
- new signals appear
- interpretations shift
- the next move changes again

This is why context cannot always be preloaded. Sometimes the act of coordinating is part of what creates the context needed for the next step.

That is also why clean summaries can mislead. They freeze what was actually dynamic, negotiated, and still evolving. The result is a familiar failure mode from earlier notes: containers and summaries start to stand in for the underlying reality.

How the Approach Varies by Setting

Not all work requires the same level of interaction, and not all settings require the same theory of context.

The right approach depends on both the nature of the work and how much interaction is needed to create meaning.

A useful shorthand is:

- in clear settings, endurants and perdurants are stable enough that context can often be preloaded, documented, or ignored

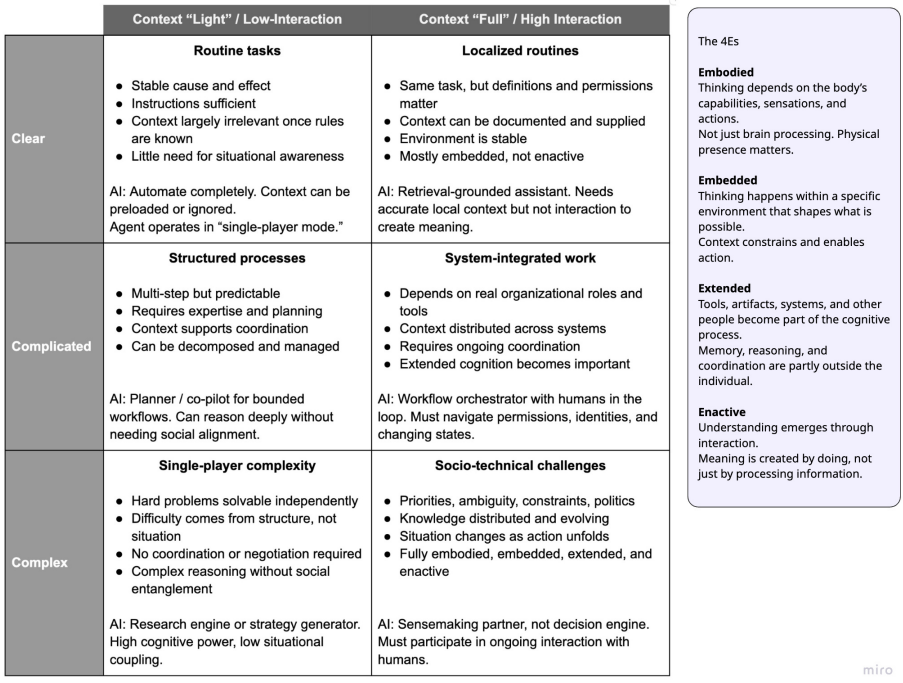

Figure 10: Matrix showing different approaches by setting, interaction level, and type of context

- in complicated settings, context may be distributed, but it can still often be aligned and coordinated across bounded structures
- in complex settings, context often emerges through interaction and cannot be separated from the work itself

That means the right use of AI also shifts:

- automate where context is stable and low-interaction
- retrieve and ground where local context matters but meaning is still portable
- support coordination where context is distributed across people and systems
- participate in shared sensemaking where understanding only forms through interaction

Reprise

It is useful to return to the original endurant-perdurant diagram here.

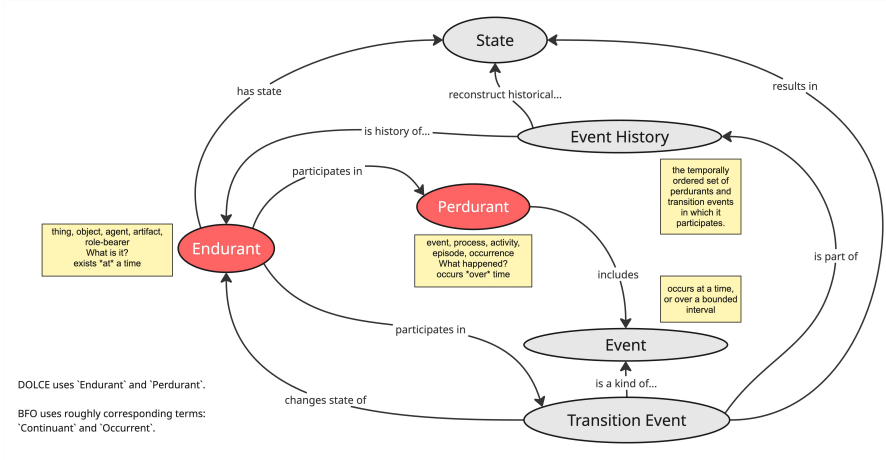

Figure 11: Endurant vs Perdurant diagram

In simple settings, this diagram can look static. There is an endurant, a process, some events, and a state. But in complex settings, the important point is where context is actually emerging.

It emerges through:

- events and transition events that change the state of an endurant
- event histories that accumulate traces of what happened

- perdurants that are still unfolding as people interact with the situation

That is why context is not just a package surrounding a message. In many complex environments, context is produced as the endurant changes state, as events accumulate into history, and as the perdurant itself continues to unfold.

Seen this way, context is partly historical, partly situational, and partly emergent. It is not just what actors bring to the interaction. It is also what the interaction produces.

The Core Insight

The practical mistake is treating every environment as if it followed the same communication model.

If you assume all understanding is transmitted, you will over-invest in summaries, pre-reads, documentation, and context packaging. If you recognize that some understanding is generated through interaction, you start to value rituals, dialogue, backbriefs, working sessions, and other ways of creating context together.

That is another way of stating the series tension:

- legibility asks how to make context portable
- métis asks how to stay close to lived reality
- endurants ask what is stable enough to name
- perdurants ask what is still unfolding and therefore must be worked through

In the simplest terms:

- Shannon asks: how do we send context well?
- 4Es asks: how do we create understanding together?

Try This Now:

- Think of one recent meeting, review, or handoff that went better after people talked live.
- Write down what was missing from the document, ticket, or summary beforehand.
- Ask: was the missing piece just more information, or did the shared understanding have to be created through interaction?

AI Opportunities (and Caveats)

Summary: AI creates real opportunities around translation, synthesis, coordination, and adaptive views, but the caveat is consistent: it can also harden bad abstractions, amplify weak data, and scale portable legibility at the expense of interaction-generated understanding.

Many recurring organizational problems become easier to see through the language of `endurants`, `perdurants`, `containers`, `anchors`, `legibility`, and `métis`.

The broad pattern is simple. Many organizational problems persist because the work is messy, the concepts are fuzzy, the views are partial, and the coordination burden is real. AI can help with many of these problems, but it can also make the same problems harder to see while spreading them further and faster.

If context is not just transmitted but often generated through interaction, then the question is not only what AI can summarize or translate. The deeper question is whether AI is helping people create shared understanding, or only packaging existing fragments more efficiently.

Seen through the `4Es`, that means asking:

- is AI improving contact with reality (`embodied`)?
- is it making the situation clearer (`embedded`)?
- is it helping cognition move across people and tools (`extended`)?
- is it supporting the interaction through which understanding emerges (`enactive`)?

At a Glance

Problem area	Real opportunity	Real caveat
Integration and translation	adapt across local variation without forcing one taxonomy	scale semantic confusion faster
Exceptions and messy work	support judgment on edge cases without full automation	over-automate the rare cases that need human sensemaking
Data quality and cognitive load	reduce maintenance burden and keep artifacts fresher	add noise, prompts, and disengagement
Recontextualization and multiple lenses	generate views for different audiences from shared sources	create subtle drift across those views
Narrative and relationships	surface patterns across scattered artifacts and signals	tell convincing stories from weak relationships
Centralization and control	support local adaptation on top of shared structures	recentralize behavior around what is easiest to measure

Translation Layers

One of the clearest opportunities is translation.

Teams regularly use the same words differently. One team's `epic` is another team's `initiative`. One system's dependency is another team's annotation. AI can help reconcile these local differences without forcing strict standardization up front.

That is valuable because many integration problems are really problems of meaning, not just data plumbing. In Shannon-style terms, AI looks helpful because it can move more context around more quickly.

But the caveat is equally important. If the underlying endurants are fuzzy and the local usage is inconsistent, AI may simply learn the mess and reproduce it. It becomes a translation layer on top of semantic drift rather than a real improvement in understanding. It makes context more portable without necessarily making it more shared. That helps the `extended` side of cognition, but may leave the `enactive` side untouched.

Messy Work and Exceptions

AI also creates opportunities around messy work and exception handling.

That matters because knowledge work is full of:

- fuzzy categories
- evolving work boundaries
- one-off anomalies
- partial exceptions that do not justify full automation

Used well, AI can help teams spot anomalies, suggest resolutions, and support semi-structured judgment without requiring everything to be turned into a rigid workflow.

The caveat is that some mess is productive. Some ambiguity helps teams think, debate, and learn. If AI smooths away the productive friction too early, it can weaken local sensemaking and push teams toward overconfidence. In 4E terms, it can interrupt the `enactive` and `embodied` work through which people discover what is actually going on.

Keeping Artifacts Alive

Another opportunity is reducing the maintenance tax on important but adjacent artifacts.

Capability maps, strategy docs, planning sheets, and other support structures often decay because the effort to keep them current competes with the actual work. AI can help by suggesting updates, flagging stale areas, and reducing the manual burden of completeness.

This is promising, but only if the prompts, suggestions, and interventions feel trustworthy. If the system creates more noise than value, people will disengage. Once that happens, the artifact still looks alive while the underlying trust disappears. The `embedded` environment appears healthy while the real cognitive support has decayed.

Multiple Lenses, One System

Many of the forever problems are really about different lenses on the same underlying reality.

- executives want coherence
- operators want actionable detail

- finance wants investment logic
- product wants outcomes and learning
- delivery teams want workable coordination

AI can help produce multiple views from shared sources, tailoring what matters for each audience without requiring endless manual recontextualization.

That is a real opportunity. But it also creates a familiar caveat from earlier notes in this series: different containers can drift away from the same underlying anchors. If AI generates a slightly different story for each audience, people may feel aligned while actually reasoning from incompatible assumptions. What looks like shared context may only be parallel interpretation.

Narratives, Relationships, and "My Dots"

AI is also well suited to work across relationships rather than just records.

That matters because real organizational understanding is rarely built from isolated objects. People connect:

- goals
- risks
- customer needs
- metrics
- dependencies
- narratives across time

AI can help surface those links, summarize across scattered artifacts, and generate "my dots" views that make the system more navigable.

But narrative power is dangerous. A coherent AI-generated story can feel true even when the underlying relationships are weak. This is another version of the legibility problem: the representation becomes compelling enough that people stop asking whether it is faithful. A generated narrative can substitute for the interaction that would have tested it. It can strengthen the `extended` artifact layer while weakening the `enactive` process that should have challenged it.

Official, Real, Ideal

One of the deepest opportunities is helping organizations navigate the gap between the official model, the real model, and the ideal model.

AI can help identify where today's process is decaying, where the real work has drifted, and where teams are compensating. It can highlight patterns that suggest what to keep, what to adapt, and what to sunset.

But if AI overfits to the current state, it can become a machine for fossilizing the present. The system gets very good at describing how things work today while making it harder to move toward a better future state.

The Series Lens

Read through the language of this series, the recurring question is not simply whether AI is useful. It is what kind of thing AI is helping us stabilize, and what kind of process it is helping us navigate.

- If AI strengthens the wrong endurants, it hardens bad containers.
- If AI hides the relevant perdurants, it removes the history and transitions needed for interpretation.
- If AI improves legibility without preserving métis, it can make the system cleaner and less truthful at the same time.
- If AI helps people work across anchors, histories, and situated interaction, it can genuinely reduce friction.
- If AI supports the `4Es` well, it can strengthen reality contact, situational awareness, distributed cognition, and shared sensemaking instead of just speeding up output.

The setting matters too:

- in clearer settings, AI can often automate or retrieve context effectively
- in complicated settings, AI can help coordinate distributed context across bounded structures
- in complex settings, AI is most valuable when it supports shared sensemaking rather than pretending to replace it

The Core Insight

AI is most useful when it helps people work with complexity rather than pretending complexity has disappeared.

The opportunity is not just better automation. It is better translation, better synthesis, better navigation across lenses, better support for local judgment, and better interfaces between messy reality and organizational legibility.

The caveat is just as important. AI can turn weak abstractions into stronger illusions. It can make bad containers look authoritative, make stale artifacts look alive, and make partial stories feel complete. When that happens, AI does not reduce the need for interaction. It merely hides that need behind more convincing outputs.

Try This Now:

- Pick one place where AI might help in your organization: translation, synthesis, status, planning, or navigation.
- Write down what weak abstraction, missing context, or fuzzy category sits underneath that use case.
- Ask: would AI help people work with the complexity there, or mainly make a shaky model look more convincing?

Twelve Practical Moves

Summary: If this guide has a practical takeaway, it is to be much more selective about what you stabilize as an `endurant`, to treat many other things as flexible `perdurants`, and to stay closer to anchors than containers.

The Key Move

Start here. The most important decision in this guide is not how to define more wrappers. It is deciding what deserves to be stabilized as a real `endurant` in your context, and what should instead be treated as a `perdurant`, something unfolding over time.

There is real power in reframing the real, durable reference points in your environment and being very selective about them. Often those are things like `teams`, `products`, `actionable inputs`, more stable focus areas, `changes released`, product taxonomies, customer segments, services, systems, and metrics. These are often better candidates for `endurants` and, in many cases, better `anchors` as well.

Candidate	Works well as an `endurant` or `anchor` when…	Watch out for…
Team	it is a real team with durable boundaries, shared context, and stable responsibility	treating temporary staffing patterns or reporting lines as if they were durable operating units

Candidate	Works well as an `endurant` or `anchor` when…	Watch out for…
Domain	it reflects a real area of responsibility, language, and coordination	drawing boundaries that look clean on slides but do not hold in practice
Milestone	it marks a real, consequential point in time that people can observe	using vague phase labels that mean different things to different groups
Release	it points to an actual change shipped or exposed to users	confusing plans, intentions, or target dates with the release itself
Actionable Input	it is concrete enough to shape decisions and behavior	turning loose signals into pseudo-objects before they are actionable
Hypothesis	it is explicit, testable, and tied to evidence or learning	using it as a euphemism for a loosely defined initiative
Goal	it is defined well enough to guide choices and tradeoffs	treating broad aspiration statements as if they were operational anchors
Service	it maps to something real in the architecture or operating model	using overloaded names that collapse multiple systems into one wrapper
Product or Service Taxonomy	it stabilizes shared language for real things people need to navigate	mistaking the classification scheme for the underlying reality
Escalation	it marks a meaningful change in attention, risk, or decision rights	turning every mention upward into the same kind of event

Candidate	Works well as an `endurant` or `anchor` when…	Watch out for…
Risk	it is specific enough to track, discuss, and act on over time	treating generic risk labels as if they carry enough context on their own

This is also why it helps to think of "flow" much more loosely in knowledge work. What you often have is not a strict cascade or a clean left-to-right timeline, but a tapestry of interlocking elements: some more work-like, some more impact-like, and some oriented around possibility, exploration, or sensemaking.

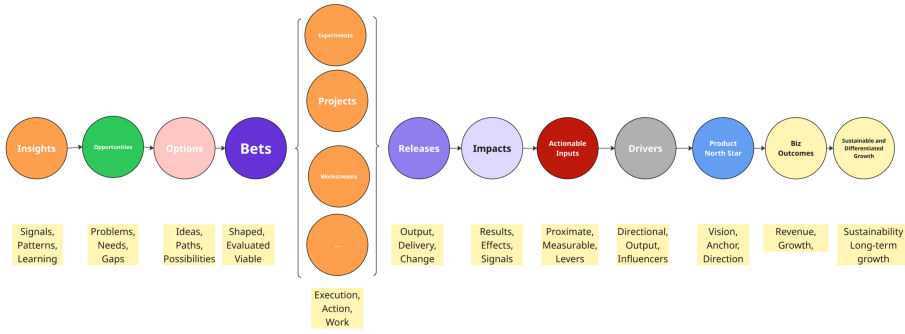

Figure 12: Loose, interlocking flow across bets, workstreams, releases, impacts, inputs, drivers, and outcomes

There is a flow here, but it is better understood as a loose ***`perdurant`*** *with many variations than as one neat cascade. Framings like observe-orient-decide-act, learn-build-measure, or diverge-converge can help a little, but the more important question is how your system actually behaves in context.*

By contrast, things like `bets`, `initiatives`, `epics`, and similar wrappers often need much more flexibility in definition. You may still name them, because organizations need names, but treat them more like `perdurants`: forms of unfolding activity, attention, coordination, and decision-making. Keep asking what is really happening over time, and keep coming back to the anchors that stay closest to reality. See Events, States, and Transitions, Practical Modeling Rule, Anchors, and Endurant Fidelity.

Eleven More Moves

- **Watch the events and exchanges, not just the wrappers.** Pay close attention to interactions, handoffs, side conversations, and the events they throw off. The point is not to standardize every shape of initiative. The point is to notice which events actually matter. See The Left of the Ticket and Why Event Storming Helps.
- **Do not force one shape of initiative where many shapes are real.** If different efforts genuinely unfold differently, forcing them into one temporal model is an anti-pattern. Make shared events legible only when they carry real significance. See Perdurant Strategy and The Containment Problem.
- **Think ritual first.** Before getting too attached to your object hierarchy or value pyramid, ask what happens when people come together, how they shape the thing around them, and what they need to see to make real decisions. See Context Through Interaction and The Core Insight.
- **Design for drill-down, not just roll-up.** People may need a high-level view, but the real test is whether they can quickly get into the relevant details and context. Build views that help people move from summary to history, anchors, and the right local reality. See How the Approach Varies by Setting and Multiple Lenses, One System.
- **Treat workarounds as data.** If people maintain parallel docs, add context in conversations, or keep re-explaining what something "really means," that is not noise. It is evidence that the official representation is too thin. See The Mismatch and Compensation.
- **Use containers for coordination, not truth.** Containers help organize work, but they become dangerous when people treat them as the primary object of understanding. When you need truth, go to anchors. See Why It Matters and Rule of Thumb.
- **Watch out for summary containers.** A green initiative, a portfolio epic, or a high-level OKR can compress multiple incompatible realities. If the upper layer is not real enough to guide action, flatten to anchors instead of stacking more wrappers. See Summary Containers and Anchors.
- **Assume scale is guilty until proven innocent.** A local model does not automatically become a portfolio model just because you moved up a layer. Test whether something is truly fractal or whether you are just manufacturing a containment hierarchy. See The Scaling Assumption, Some Things Are More Fractal, and

Scale of the Local.

- **Standardize interfaces and signals before you standardize everything else.** Shared milestones, decision points, handoffs, and input metrics often travel better than one rigid workflow or one universal taxonomy. See What To Standardize and Endurant Fidelity.
- **Read the organizational context before prescribing the model.** Look at coupling, the balance between legibility and `métis`, and the underlying factors shaping both before you standardize a workflow, taxonomy, or operating cadence. See The Cycle, At a Glance, and Coupling Reality vs. Coupling Abstraction.
- **Use AI to translate and surface, not to finalize meaning.** AI is strongest when it helps people compare views, navigate ambiguity, and surface patterns. It is weakest when it makes weak abstractions feel settled and complete. See Translation Layers, Messy Work and Exceptions, and The Series Lens.

Try This Now:

- Write down three things in your organization that deserve a more stable name because they are real anchors.
- Then write down two things your organization treats like stable objects that would be better understood as unfolding activity.
- Ask: if you had to redesign one status view tomorrow, which anchors would you bring closer to the surface first?

Glossary

Summary: A working glossary for the series. These definitions aim for practical usefulness, not false precision. Use them to stabilize the language of the guide while still leaving room for local variation.

How to Use This Glossary

This glossary is meant to help you read the series and reuse the ideas in your own setting. The terms here are intentionally operational. They are not trying to be the last word in formal ontology.

Some of these terms are durable enough to carry across notes with roughly the same meaning. Others change shape depending on the context. When that happens, the short definition here should help orient you, and the linked note should supply the nuance.

At a Glance

Term	Short meaning
`anchor`	A durable reference point tied closely to observable reality.
`container`	A grouping wrapper used to organize, coordinate, or summarize work.
`context`	The history, situation, relationships, and interpretation needed to make sense of what is happening.

Term	Short meaning
`coupling`	The degree to which parts of the system depend on one another to move effectively.
`dynamic optimization`	Treating the work as search, learning, and adaptation over time.
`endurant`	A relatively stable thing you can point to at a moment in time.
`endurant fidelity`	How well your chosen durable objects match the reality of the work.
`event`	A bounded occurrence inside a larger unfolding process.
`Event Storming`	A way of reconstructing the real history, decisions, actors, and state changes behind visible artifacts.
`legibility`	A simplified, portable representation that makes a system easier to see and manage.
`métis`	Local, situated practical judgment that helps people make things work in context.
`perdurant`	Something that unfolds over time and only makes sense across a history or sequence.
`perdurant strategy`	How you choose to model time, rhythms, and process variation.
`RAG status`	A compressed red-amber-green signal whose usefulness depends on the quality of the model underneath it.
`real SDLC(s)`	The many actual paths work takes in practice.
`scale of the local`	How much of the surrounding system a team must understand to act well.
`state`	The condition of an `endurant` at a point in time.

Term	Short meaning
`static optimization`	Treating the work as if the objective, constraints, and solution can be fixed early.
`summary container`	A roll-up wrapper that compresses many realities into one signal.
`theoretical SDLC`	The official or standard lifecycle people use to talk about work.
`transition`	A meaningful change in state, usually best modeled as an event.

Core Ontology Terms

Anchor

An `anchor` is a kind of `endurant` tied closely to observable reality. It stays close to what actually changes in the world: a team, a service, a metric, a production change, a customer segment, or another durable point of reference. Anchors are where you look when you need to understand reality rather than merely organize it. See Containers vs. Anchors and Twelve Practical Moves.

Container

A `container` is a grouping wrapper used to organize work, coordinate attention, or summarize effort. `Initiatives`, `projects`, `epics`, and similar wrappers often work this way. Containers are useful, but they are easy to mistake for the underlying reality, especially when their contents drift over time. See Containers vs. Anchors, Theoretical SDLC vs. Real SDLC(s), and Will It Scale?.

Container-anchor misalignment

`Container-anchor misalignment` happens when the wrapper says one thing and the closer-to-reality anchors say another. The initiative looks green, for example, while the service, team, or metric underneath it is clearly in trouble. See Containers vs. Anchors.

Endurant

An `endurant` is something treated as wholly present whenever it exists. In this series, that usually means a durable reference point such as a `team`, `service`, `release`, `metric`, or another thing whose identity can persist while its properties change. See Endurant vs. Perdurant, From Clear Flows to Complex Systems, and Twelve Practical Moves.

Event

An `event` is a bounded occurrence inside a larger unfolding story. Events matter because they preserve what changed, when, and often why. When the history matters, events usually belong in the model more than a static label does. See Endurant vs. Perdurant and Case Study: Event Storming Perdurants.

Missing perdurant

A `missing perdurant` is what happens when a model preserves the durable object but loses the unfolding story around it. You still have the item, but not the sequence, transitions, loops, and interpretation needed to understand its condition. See RAG Status, Endurants, and Perdurants.

Nouns versus verbs

`Nouns versus verbs` is a reminder that many formal nouns in organizations are really shorthand for ongoing activity. An `initiative`, `objective`, or `epic` often points at a process of coordination and interpretation more than at a neat object. See Case Study: Event Storming Perdurants and Twelve Practical Moves.

Perdurant

A `perdurant` is something that unfolds through time. It is made legible through its phases, history, transitions, and events rather than by a single snapshot. In practice, many things organizations name as if they were stable objects are better understood as `perdurants`: unfolding activity, coordination, negotiation, and learning over time. See Endurant vs. Perdurant, Case Study: Event Storming Perdurants, and Twelve Practical Moves.

State

`State` is the condition of an `endurant` at a point in time. A machine can be running or stopped. A service can be degraded or healthy. A team can be overloaded or stable. State helps describe the condition of the durable thing, while the change between states is better modeled as an event. See Endurant vs. Perdurant.

Summary Container

A `summary container` is a higher-level roll-up wrapper that compresses many different realities into one headline signal. Portfolio status, theme status, and quarterly roll-ups often behave this way. They can help leaders navigate, but they often strip away exactly the context needed for correct interpretation. See RAG Status, Endurants, and Perdurants.

Temporal parts

`Temporal parts` are the phases or segments that make up a `perdurant`. They matter because they let you describe an unfolding thing in terms of before, during, after, and the meaningful pieces in between. See Endurant vs. Perdurant.

Tight endurant-perdurant mapping

`Tight endurant-perdurant mapping` means the stable thing being tracked lines up well with the unfolding work around it. This is one reason some operating metrics and statuses feel truthful while others feel thin or misleading. See RAG Status, Endurants, and Perdurants.

Time-slice or snapshot test

The `time-slice` or `snapshot test` is a practical modeling question: if you freeze time, do you still have the whole thing, or only one moment in its unfolding? If the whole thing is present, you are probably looking at an `endurant`. If the meaning depends on the sequence, you are probably looking at a `perdurant`. See Endurant vs. Perdurant.

Transition

A `transition` is a meaningful shift from one state to another. In this guide, transitions are important because they often preserve the real operational story better than snapshot categories alone. See Endurant vs. Perdurant and RAG Status, Endurants, and Perdurants.

Wrong endurant

A `wrong endurant` is a thing the organization has decided to treat as stable even though it is too fluid, overloaded, or internally inconsistent to bear that role well. Once that happens, reporting gets cleaner while understanding gets worse. See RAG Status, Endurants, and Perdurants and Factors Shaping Legibility and Métis.

Representation and Sensemaking

4E cognition

`4E cognition` is shorthand for embodied, embedded, extended, and enactive views of cognition. In this series, it matters because it supports the idea that understanding is often generated through participation, tools, rituals, and interaction, not just by reading a packet. See Context Is Not Just Transmitted and AI Opportunities (and Caveats).

Action and understanding co-evolve

`Action and understanding co-evolve` means people often do not fully understand first and act second. In complex settings, action changes the situation, which changes the signals, which changes the interpretation. Understanding is part of the process, not always a prerequisite to it. See Context Is Not Just Transmitted.

Compensation

`Compensation` is the extra work people do when the official model is too thin to support the work. Parallel spreadsheets, repeated explanations, side channels, and hand-built tracking are often not redundancy for its own sake. They are attempts to rebuild lost context. See Métis vs. Legibility and Twelve Practical Moves.

Context

`Context` is not just background information. It includes relevant history, state changes, relationships, constraints, timing, interpretation, and the interaction needed to produce shared understanding. In complex settings, context often has to be generated together rather than merely transmitted. See Context Is Not Just Transmitted, Case Study: Event Storming Perdurants, and AI Opportunities (and Caveats).

Legibility

`Legibility` is the drive to make a system visible, readable, portable, measurable, and governable. Legible representations are not inherently bad. The problem begins when the simplified representation gets mistaken for the whole reality. See Métis vs. Legibility, Factors Shaping Legibility and Métis, and Will It Scale?.

Legibility sufficiency test

The `legibility sufficiency test` asks whether the artifact can stand on its own without constant translation from insiders. If a newcomer cannot make sense of it, or if the real meaning only lives in surrounding conversations, the artifact is not doing as much work as it appears to be doing. See Métis vs. Legibility.

Métis

`Métis` is local, situated practical judgment. It is the know-how people develop by working in the real setting, not just by reading the official representation of it. Métis matters most where the categories are unstable, the work is changing, and actors must interpret in motion. See Métis vs. Legibility, Coupling, Legibility, and Métis, and Factors Shaping Legibility and Métis.

Multiple lenses, one system

`Multiple lenses, one system` refers to the need to view the same reality differently for executives, operators, product, finance, and others without inventing separate worlds. The challenge is to support different views while staying anchored to the same underlying reality. See AI Opportunities (and Caveats).

Official, real, and ideal models

The `official model` is how the organization says things work. The `real model` is how they actually work. The `ideal model` is how people think they should work. Useful diagnosis often starts by comparing all three rather than pretending one of them is the whole story. See AI Opportunities (and Caveats).

Portable versus interaction-generated context

`Portable context` is context you can package and move with relatively little loss. `Interaction-generated context` is understanding that only emerges through working, talking, and interpreting together. The distinction matters because many organizations try to solve the second problem with tools built for the first. See Context Is Not Just Transmitted.

Shannon-style communication

`Shannon-style communication` treats communication as the transmission of a message from one party to another, with success depending on correct encoding and decoding. That model can help in stable settings, but it is too thin for situations where understanding emerges through interaction. See Context Is Not Just Transmitted.

System-métis mismatch

A `system-métis mismatch` appears when formal systems assume stable categories, clear boundaries, and portable meaning while the real work depends on local judgment and ongoing interpretation. Much of the friction in operating systems comes from this mismatch. See Métis vs. Legibility.

Translation layers

`Translation layers` are structures, people, or tools that help reconcile differences in vocabulary, model, or perspective across groups. They can be genuinely useful, but they can also mask unresolved disagreement if they make mismatch look more settled than it is. See AI Opportunities (and Caveats).

System Conditions and Design

Abstraction quality

`Abstraction quality` is how well the shared model compresses reality without becoming misleading. Good abstractions reduce friction while still supporting local judgment. Poor abstractions increase translation work and brittle coordination. See Coupling, Legibility, and Métis.

Command Towers

`Command Towers` describes a regime of high legibility and centrally imposed structure. It can create a strong sense of visibility, but it often suppresses local judgment and slows adaptation because so much depends on the shared model being updated from the center. See Coupling, Legibility, and Métis.

Constraint strategy

`Constraint strategy` is where and how control is applied in the system. A better strategy usually relies on fewer, more reliable anchors and decision points rather than trying to make everything equally visible and equally governed. See Factors Shaping Legibility and Métis.

Coupling

`Coupling` is the degree to which parts of a system depend on one another to move well. Higher coupling usually increases the need for better abstractions, richer context, more translation, or more local judgment. It is one of the main reasons the same operating model works well in one setting and fails in another. See Coupling, Legibility, and Métis and Factors Shaping Legibility and Métis.

Coupling reality versus coupling abstraction

`Coupling reality versus coupling abstraction` separates dependencies that are genuinely in the work from dependencies introduced by the model, planning wrapper, or coordination structure. This distinction matters because some coordination pain is real and some of it is self-inflicted. See Factors Shaping Legibility and Métis.

Endurant Fidelity

`Endurant fidelity` is the degree to which the durable things in your model match the durable things in reality. If the wrong object becomes the official reference point, the system accumulates translation work and false confidence. See Factors Shaping Legibility and Métis and Twelve Practical Moves.

Federated Islands

`Federated Islands` describes a low-legibility, high-métis environment where local groups can work effectively on their own terms but shared

understanding across the system is weak. It often feels better locally than a locked grid, but global coordination becomes fragile. See Coupling, Legibility, and Métis.

Hands-Off Gridlock

`Hands-Off Gridlock` describes a system with weak shared structure and too much coordination burden crossing boundaries. No one is truly in control, but the lack of stabilizing abstractions does not create freedom so much as perpetual negotiation. See Coupling, Legibility, and Métis.

Locked Grid

`Locked Grid` describes a system with high coupling and weak abstractions, where work is forced through rigid structures that do not match reality. It is one of the most brittle states because both legibility and métis are constrained. See Coupling, Legibility, and Métis.

Loose perdurant flow

`Loose perdurant flow` is a way of describing knowledge work without pretending it behaves like one neat left-to-right cascade. There may still be a flow, but it often looks more like interlocking threads of work, impact, possibility, and sensemaking. See Twelve Practical Moves.

Perdurant Strategy

`Perdurant strategy` is how you choose to model rhythms, cadences, histories, and variations in how work unfolds. Some systems can tolerate one dominant cadence. Others need multiple overlapping rhythms and cannot honestly be reduced to a single clean flow. See Factors Shaping Legibility and Métis, From Clear Flows to Complex Systems, and Twelve Practical Moves.

Scale of the Local

The `scale of the local` is how much of the surrounding system a person or team must understand to act well. Sometimes the local really is small. Sometimes the local is already large, which means autonomy cannot eliminate coordination work. See Factors Shaping Legibility and Métis.

Operational Frames and Practices

AI

`AI` is treated here as a tool that can translate, summarize, connect, and support navigation across messy systems. It can help, but it can also harden bad abstractions, over-compress real variation, and create false confidence if the underlying model is weak. See AI Opportunities (and Caveats). ### Blast radius {#glossary-blast-radius}

`Blast radius` is the downstream effect a signal, decision, or change has across the surrounding system. In this series it helps explain why a small shift in one place can produce large invisible coordination work elsewhere. See Case Study: Event Storming Perdurants.

Containment problem

The `containment problem` is what happens when organizations keep adding higher-level wrappers so work can be summarized cleanly upward. Those wrappers can improve navigability while degrading fidelity. See Will It Scale?.

Dynamic optimization

`Dynamic optimization` is a way of framing work where the objective evolves, the constraints can be reshaped, and the solution is discovered over time rather than computed upfront. In this series it fits product work because the important endurants are often still being clarified, the perdurant is still unfolding, and learning changes the target as you go. See Static vs. Dynamic Optimization, Perdurant strategy, Constraint strategy, and Endurant fidelity.

Event Storming

`Event Storming` is a way of reconstructing the real history of a domain by surfacing actors, events, decisions, information changes, and dependencies across time. In this series it is especially useful because it restores the relationship between visible artifacts and the richer perdurants behind them. See Case Study: Event Storming Perdurants.

Fractal structures

`Fractal structures` are patterns that can repeat across levels without becoming meaningless, but only when they remain anchored to local real-

ity rather than copied mechanically. The lesson is not "reuse everything," but "reuse only what preserves fidelity across scales." See Will It Scale?.

Left of the ticket

`Left of the ticket` refers to the messy, interpretive, relational work that happens before a durable artifact appears in a tool. It matters because organizations often mistake the first visible row in a system for the beginning of the real story. See Case Study: Event Storming Perdurants.

Organism, not bigger cell

`Organism, not bigger cell` is a reminder that scaling changes the nature of the system. Larger systems develop coordination needs and properties that are not just enlarged versions of the smaller case. See Will It Scale?.

Promotion versus reality

`Promotion versus reality` names the moment a messy evolving situation becomes a durable official artifact and then starts to be treated as the story itself. The promoted object is useful, but it is rarely the full history. See Case Study: Event Storming Perdurants.

RAG status

`RAG status` is a red-amber-green signal used to summarize current condition. It works best when it sits on top of stable endurants, visible transitions, and intelligible perdurants. It breaks down when the tracked object is vague, the unfolding story is invisible, or the roll-up container hides conflicting realities. See RAG Status, Endurants, and Perdurants and Containers vs. Anchors.

Real SDLC(s)

`Real SDLC(s)` are the many actual paths work takes in practice. Different work varies by uncertainty, coordination load, time horizon, decision structure, risk, and value timing. That is why one official lifecycle is often too blunt to describe what is really happening. See Theoretical SDLC vs. Real SDLC(s).

Scale

`Scale` in this guide is not just "more of the same." It changes what counts as local, what needs coordinating, what abstractions can hold, and whether the same terms still refer to the same kind of thing. See Will It Scale?.

Scaling assumption

The `scaling assumption` is the belief that the same labels, structures, and flows can be stretched upward without changing kind. Much of the pain around organizational design comes from discovering that what worked locally no longer names the same reality at larger scale. See Will It Scale?.

Static optimization

`Static optimization` is a way of framing work where the objective, constraints, and solution are treated as knowable early enough to plan around directly. It can be useful in clearer, more stable systems, but it becomes misleading when teams apply it to work whose goals, constraints, and meaning shift during execution. See Static vs. Dynamic Optimization, What to standardize, and Perdurant strategy.

Theoretical SDLC

The `theoretical SDLC` is the official or standard lifecycle people use to describe how work should move. It is often useful as a coordinating abstraction, but it is still a model. Problems begin when the model is treated as the whole reality. See Theoretical SDLC vs. Real SDLC(s).

What to standardize

`What to standardize` is the practical design question that matters more than whether everything should follow one official process. The series generally favors standardizing interfaces, milestones, decision points, and risk signals more than imposing one universal path. See Theoretical SDLC vs. Real SDLC(s).