THE WORKSHOP

Cynefin Sensemaking

A 75-minute pattern for sorting a pile of problems into the kinds of actions they reward. Clear, complicated, complex, chaotic, confusion — and the realisation that disagreement about which domain a problem belongs in is often the problem itself.

2027-01-08
barkingiguana.com/writing/the-workshop-cynefin-sensemaking/

A facilitator's playbook — print, annotate, and keep on the table.

Contents

Cynefin Sensemaking 3
What's It For 3
What It's Not For 4
Definitions & Background 5
Inputs 6
Outputs 7
Who's Needed 7
How To Run It 8
What Can Go Wrong 12
Next Steps 12
Variants 13

Not every problem needs a workshop. Cynefin is a sensemaking framework: you make sense of where each problem sits across five domains, so you reach for the right level of analysis instead of running a three-hour Event Storm on something three people can sort in a corridor. Worked example: Not Everything Needs a Workshop.

Cynefin Sensemaking

Cynefin Sensemaking sorts a pile of open problems across five domains (clear, complicated, complex, chaotic, confusion) and attaches an action paradigm to each, so the team reaches for the response shape the problem actually rewards rather than the one its toolbox is built for. Invented by Dave Snowden at IBM in the late 1990s and developed through the Cynefin Centre since; sometimes called the Cynefin framework or the five domains, and often confused with risk assessment, which it isn't. In Snowden's terms it's a *sensemaking* framework, not a *categorisation* one: domains emerge from the problems, not the other way round.

At a glance

- *Who, for how long:* a facilitator who knows the domain definitions cold, a delivery lead, an engineering lead, a product lead, and an ops or CS voice. Four to six people, seventy-five minutes.
- *What you walk out with:* a populated canvas with 8-20 problems placed by domain, an action paradigm and a first concrete action per problem with owners and dates, a list of contested placements, and a revisit booked two weeks out.
- *When to reach for it:* the team has a running list of problems and keeps applying one shape of response to all of them, or retros keep producing actions that don't stick. Not for a single problem (there's nothing to compare), not for problems that aren't articulated yet, and not as post-hoc cover for a decision someone's already made.

What's It For

A delivery lead runs a retro about an outage. The team writes five-whys (the technique of asking "why?" five times to chase a cause back through its layers) identifies a root cause, commits to a fix, and goes home satisfied. Three weeks later, a superficially similar outage happens. The root cause is different. The fix didn't apply.

Five-whys works beautifully for complicated problems: ones where the cause is knowable with enough investigation. It's the wrong shape of tool for complex ones, where the causes are only visible in retrospect, where the system's behaviour emerges from interactions rather than being determined by any one component. The outage wasn't caused by the thing the five-whys identified; it was caused by a set of pressures that produced the thing the five-whys identified, and also the next thing, and the next. Running five-whys on a complex problem produces a tidy story that happens to be wrong.

Cynefin exists because teams collapse all their problems into "problems" and then reach for the same set of tools. Some problems reward careful analysis: the answer can be found. Some reward safe-to-fail experiments (a small probe whose downside is contained even if it goes wrong): the answer has to be grown. Some reward expert judgement: the answer is knowable by someone, just not by everyone. And some reward immediate stabilisation: the answer is *later*; the first action is to stop the bleeding.

The workshop is a forcing function to have that conversation explicitly, about real problems the team is facing, in time to change how they're being handled. Confusion, the central domain, sometimes called *aporia* (the honest "I don't yet know which kind of problem this is"), is a feature, not a bug. A problem you can't yet place is a signal to break it down, not to pretend you know.

Disagreement about which domain a problem belongs in is often *the* problem. When a product lead treats something as complicated ("we just need to analyse this") and an engineer treats it as complex ("we don't know what we don't know here"), the argument about the fix is really an argument about the domain. Cynefin makes that argument explicit.

Reach for it when:

- The team has a running list of problems and no shared view on how to approach them
- Retros keep producing actions that don't stick, suggesting the problem shape is being misread
- A new initiative spans work of clearly different kinds (known fixes, research spikes, crises, and greenfield) and the team is applying one approach to all of them
- Two people disagree persistently about how to handle something and the disagreement feels underneath the words
- A recent incident is about to drive a process change and you want to check whether the change is in the correct domain

What It's Not For

Skip it when:

- The team has one problem, not several. Cynefin is comparative; one problem doesn't need a workshop.
- The problems aren't articulated yet. Spend the hour writing them down instead.
- Cynefin is being used to justify a decision someone's already made. The session becomes a post-hoc rationalisation and is worse than no session.
- The room can't hold open disagreement. Cynefin requires people to contest domain placement, which is easier to say than to do.

Trade-offs to weigh before you run it:

Benefits. A shared map of the problem space, with matching action paradigms attached. A stated "we don't know yet" list that doesn't embarrass anyone, because confusion is a valid placement. Recognition that the team's preferred tools match a subset of its problems. Disagreements about approach surfaced as disagreements about domain, which are easier to resolve. A follow-up cadence (revisit in two weeks) that lets emergent data from complex-domain probes update the map.

Costs. 75 minutes of 4-6 people. Emotional cost when the session reveals the team has been running complicated-domain tools on complex-domain problems for months. Ongoing cost: Cynefin is a live map, not a one-time artefact; revisits are part of the pattern. The jargon tax: the domain names take a few sessions to feel natural.

Failure modes. Placements get made, actions get assigned, and nothing changes because the team reverts to its old tools by Monday. Complex problems get complicated-domain actions (*"we'll analyse this"*) because the team doesn't know what a probe looks like. Confusion is treated as failure, so nothing lands there, so the session loses its most honest placements. The framework is used to justify decisions already made rather than to think. The senior voice's domain placement becomes the official one regardless of evidence.

Stop signals. The team can't articulate a safe-to-fail probe for any complex problem; the action paradigm will not be used. Everyone places everything in the same domain; either the session is broken or the team is in a narrow state. The session is being run post-hoc to justify an action; end and reschedule when the decision isn't pre-made.

Stopping and re-briefing the domain distinctions is not failure. Producing a canvas full of complicated-shaped actions for complex-shaped problems is.

Definitions & Background

Cynefin is a *sensemaking* framework, not a *categorisation* framework: the distinction matters. Categorisation says "I have a frame, where does this problem fit?" Sensemaking says "I have a problem, what kind is it?" Categorisation assumes the framework precedes the data; sensemaking lets data and framework co-evolve. The domains emerge from the problems, not the other way round. The version that matters for teams has five: four with defined action paradigms, plus the central domain of confusion where you haven't yet decided which of the other four applies. One paragraph each; enough to land the session.

Clear (obvious). *Cause is known and the fix is known.* Cause and effect are direct; best-practice exists. Action paradigm: sense – categorise – respond. See it, recognise its type, apply the known fix. Examples: *"the build is failing because a dependency pin drifted," "the subscriber signed up but the welcome email didn't send."* The risk in this domain is complacency, which tips Clear straight into *Chaotic*, not into Complicated. Treat a clear problem casually and the system fails catastrophically; the boundary between Clear and Chaotic is a cliff, not a slope.

Complicated. *Cause is knowable in advance with enough analysis.* Good-practice rather than best-practice, multiple valid answers. Action paradigm: sense – analyse – respond. Experts and investigation get you there. Examples: *"our database queries are getting slower and we don't know why yet," "the payment integration fails for 3% of subscribers and the pattern isn't obvious."* Most engineering work lives here; most tools in the team's toolbox are shaped for here.

Complex. *Cause is only knowable in retrospect.* The system's behaviour emerges from interactions. Action paradigm: probe – sense – respond. Run safe-to-fail experiments, watch what emerges, amplify what works, dampen what doesn't. Examples: *"subscriber churn is up this quarter and we don't know why," "the team's morale has been sliding and the interventions aren't sticking," "we keep having near-misses during Friday pack-out but no single cause."* Running complicated-domain tools here produces the *tidy-story-that's-wrong* outcome.

Chaotic. *Cause is unclear and the system is unstable.* No time for analysis or probes. Action paradigm: act – sense – respond. Stabilise first, understand later. Examples: *"the site is down," "the founder has just quit and nobody knows what happens Monday," "a vulnerability was disclosed and exploit attempts are already on the logs."* The pattern is: stop the bleeding, then return to the less-urgent domains.

Confusion (or *aporia*). The central domain. You don't yet know which of the other four applies. Action paradigm: break the problem down. Split it until each piece has a clear home. Confusion comes in two flavours, and they need different responses:

- *Aporetic confusion*: the honest "I don't know which domain this lives in." Productive. The answer is to break the problem down until each piece has a home. Teams that skip this step place everything as complicated by default.
- *Confused confusion*: false certainty. People are sure where the problem belongs, but they don't agree, and they're each acting accordingly. Dangerous. The first move is to surface the disagreement; the second is to admit that nobody knows yet.

Example: *"the project is struggling"* decomposes into *some parts that are complicated* (we don't know the root of the perf issue), *some parts that are complex* (team dynamics), *some parts that are clear* (we forgot to renew a certificate). The response differs per piece.

Quick reference: which domain?

Domain	Discriminator	Action paradigm
Clear	Cause is known. Fix is known.	sense – categorise – respond
Complicated	Cause is knowable in advance with enough analysis.	sense – analyse – respond
Complex	Cause is only knowable in retrospect.	probe – sense – respond
Chaotic	Cause is unclear and the system is unstable.	act – sense – respond
Confusion	We can't yet tell which of the other four applies.	break it down

The single sharpest test: *if a very good engineer analysed this for a week, would they find the cause?* Yes → complicated. Maybe → probably complex. Definitely no, you only learn the cause by changing the system → complex.

Inputs

- A list of 8–20 problems the team is currently dealing with. Each problem written as a short sentence. No pre-sorting.
- A flat wall or whiteboard you can draw the Cynefin canvas on: four quadrants (clear, complicated, complex, chaotic) with a central region (confusion).
- Sticky notes, one problem per note.
- Domain cards and action cards for the brief and the response phase.
- A rough shared understanding of the framework's five domains: brief beforehand or as Phase 1.

Outputs

What lands at the end of the session:

- A populated Cynefin canvas, with each problem placed in a domain (or honestly placed in confusion), and the disagreements that surfaced during placement still visible.
- An action paradigm per problem (sense-categorise-respond, sense-analyse-respond, probe-sense-respond, act-sense-respond, or break-it-down) attached to a first concrete action.
- An owner and a date for each action. Probes have metrics and decision rules; analyses have analysts and time-boxes; breakdowns have return dates.
- A list of contested placements: problems where the room disagreed about the domain, sometimes resolved by time-boxed treatment ("we'll treat it as complicated for two weeks; if analysis hasn't produced an answer, we'll treat it as complex").
- A revisit date, typically two weeks out, when the canvas comes back to the team and emergent data updates the map.

Photograph the canvas in panoramic, then each domain in close-up.

These outputs feed straight into:

- Impact Mapping. Impact Mapping gives you candidate deliverables; Cynefin tells you which action paradigm each one rewards. Complex-domain deliverables want probes, not full builds.
- Assumption Mapping. Probes (the complex-domain action) and assumptions (what Assumption Mapping catches) are the same shape. Use Assumption Mapping to design the probe once Cynefin has identified the domain.
- Wardley Mapping. Wardley tells you where a component sits in its evolution; Cynefin tells you what action shape the problem rewards. Composes for strategy: a genesis-stage component almost always carries complex-domain problems.
- Decision Tables. Decision tables are a clear-domain tool. A complex problem with a decision table is either being misplaced or being prematurely collapsed.
- Ensemble Programming. Ensemble is the probe-and-sense tool when the complex problem lives in code. A whole team working in a complex codebase will sense emerging patterns faster than any individual analyst.
- Example Mapping. Example Mapping is a complicated-domain tool. A story that resists Example Mapping (red cards piling up, rules that won't stabilise) is often a complex-domain story pretending to be complicated.

Who's Needed

Four to six people, seventy-five minutes:

- Facilitator. Ideally someone who has used Cynefin before, because the domain definitions are easy to slide past and the workshop depends on them landing. If you haven't run it before, read Snowden's canonical material and work through a handful of placements privately first.
- Delivery lead. The person who's going to have to change how work is being run if the domain placements reveal a misfit. Mandatory; without them, the placements are recommendations.

- Engineering lead. Most of the team's complex problems live in their head, often as frustrations with tools that don't fit the shape of the work.
- Product lead. The problems-with-subscribers and problems-with-direction domains are theirs, and they're usually the person pushing for complicated-shaped responses to complex-shaped problems.
- Ops / CS voice. The complex-domain specialists in every team, though they rarely use the language. They know which problems have never responded to analysis.
- Domain experts (optional, by problem). If a particular problem needs a deep expert's view, invite them specifically for that problem. They don't need to stay the whole session.

Above six and the argue-the-contested-ones phase fragments. Below four and there aren't enough perspectives to produce disagreement.

Who to leave out:

- People who will re-open the domain definitions mid-session. Philosophical debate about Cynefin itself is a different conversation; gate it before the session or after, not during.
- Senior stakeholders who will use the placements to pre-judge people. Cynefin placements sometimes imply that someone has been handling a problem wrongly. That has to be a safe conversation in the room.
- Large observer groups. Observers change the honesty of the placement. Keep the room small.

How To Run It

Phase	Duration	Materials	Key question
Brief the five domains	10 min	Canvas, domain cards	"What are the action paradigms?"
Dump problems	10 min	Sticky notes	"What's on the list?"
Place problems on the canvas	20 min	Canvas, problems	"Where does each belong?"
Argue the contested ones	15 min	Contested pile	"Why does it belong there and not there?"
Choose an action paradigm	15 min	Action cards	"How will we respond to this?"
Wrap up and commit	5 min	Owners + dates	"Who owns what next?"
Total	~75 minutes		

The session's rhythm: quiet brief, rapid dump, quick placement, slow argument, decisive response. The argument phase is where the value lands; don't let the placement phase eat its time.

Phase 1. Brief the five domains (10 min)

Draw the canvas: four quadrants (clear, complicated, complex, chaotic) with a central region (confusion). Walk through each domain with one sentence of definition and one example.

> *"Five domains. Clear: we know what's going on, we apply the known fix. Complicated: we don't know yet, but an expert can work it out with enough time. Complex: we can't know in advance; we have to run experiments and watch what happens. Chaotic: things are actively going wrong, stabilise first. Confusion: we haven't decided which of the other four this problem belongs in, so we're going to break it down."*

Give one example per domain that the room recognises. Then make one thing explicit:

> *"The framework doesn't grade problems by difficulty. A complex problem isn't harder than a complicated one; it's differently shaped. Applying complicated-domain tools to a complex problem is how retros keep producing actions that don't stick."*

What to watch for:

- Collapsing complicated and complex. Most teams read them as synonyms. The distinction is whether the cause is knowable in advance: complicated yes, complex no. Drill it until the room can tell the difference.
- The chaotic-is-everywhere reflex. Someone wants to place everything as chaotic. Usually the team has been in crisis too long. Quietly move on; the placement phase will correct it.
- Philosophical debate. *"Isn't everything complex at some level?"* Technically yes; practically no. The framework is a working tool, not a theory of systems. Park the philosophy.

Phase 2. Dump problems (10 min)

Each participant writes problems on sticky notes. One problem per note. Silent, individual, rapid.

> *"What are the open problems on your mind right now? Things you've been stuck on, things you've been arguing about, things the retros keep re-surfacing, things you're worried about. One per note. No filtering."*

Expect 15–30 notes for a team of five. If you get fewer than 10, the team is holding back or this workshop is the wrong fit. If you get more than 40, the problems aren't problems; they're items. Cluster before placement.

What to watch for:

- Generic notes. *"Communication."* Not a problem; a topic. Push: *"What's the specific communication problem you're seeing?"*
- Problems disguised as solutions. *"We need a standup."* Reframe: *"What's the problem a standup would solve?"*
- Self-censoring. The most useful problems are often the ones people don't want to write. Anonymous writing can help; name it as an option.

Phase 3. Place problems on the canvas (20 min)

The team places each note on the canvas. First pass is silent: people place problems they feel confident about. Second pass is open: one person picks up a note and says where they'd place it and why; the room challenges or confirms.

> *"If you're sure where a problem belongs, place it. If you're not sure, put it in confusion in the middle; that's an honest placement, not a failure."*

Good placement is about the *shape of the response the problem rewards*, not about how difficult it is. A clear problem can be big and expensive (a migration with a known playbook); a complex problem can be small and cheap (a quirky team dynamic that comes and goes).

The central confusion area earns its keep. If the team places nothing there, push: *"Every team has at least one problem they can't currently classify. What's ours?"*

What to watch for:

- Everything as complicated. The default placement for teams that have been burned or teams overloaded with expertise. Challenge: *"If a very good engineer analysed this for a week, would they find the answer?"* If the honest answer is *"maybe,"* it's probably complex.
- Domain-shopping. Someone places a problem in whichever domain justifies the response they want. Name it: *"You're placing this as chaotic because you want to act immediately. Is it actually chaotic, or is it complex and we want to do the probe-sense-respond work?"*
- Refusing to use confusion. The team places everything definitively. Almost never honest. *"Anything we're not sure about goes in the middle. That's a valid placement."*
- Ignoring the clear quadrant. Teams often skip clear because it feels beneath them. Problems that belong there belong there; they don't need workshops, they need playbooks.

Phase 4. Argue the contested ones (15 min)

For each note where two people disagree about placement, open a short conversation.

> *"This problem is placed in complicated and two people think it's complex. I'd like one person from each side to say why."*

The disagreement is almost always more informative than the eventual placement. A problem is placed as complicated because someone believes the cause is findable; it's placed as complex because someone believes the system's behaviour is emergent. Making the beliefs explicit reveals which tools the team has been reaching for and why they haven't worked.

Resolution isn't always consensus. Sometimes the problem is best placed *between* two domains, with the action paradigm explicitly chosen from one of them with a time-box: *"We'll treat it as complicated for two weeks, and if analysis hasn't produced an answer, we'll treat it as complex."* That's a legitimate answer and sometimes the correct one.

What to watch for:

- The senior-voice default. Disagreements resolve in the senior person's direction even when the other voice is more informed. Ask the quieter voice directly.
- False consensus. The person who placed the note capitulates without being convinced. Stay with it: *"Are you actually persuaded, or are you letting it go?"*
- The disagreement that's actually a problem statement. Two people disagreeing about the domain are often describing two different problems that share a sticky note. Split it.

Phase 5. Choose an action paradigm (15 min)

For each placed problem, the room names the action paradigm and the first concrete action.

- Clear problems: *"Apply the playbook. Who owns it?"*

- Complicated problems: *"Who analyses, with how long a time-box?"*
- Complex problems: *"What's our first safe-to-fail probe? How will we know if it worked?"*
- Chaotic problems: *"What stabilises it now? Who's doing that?"*
- Confusion problems: *"Who breaks it down? When does it come back to the team?"*

The outputs are actions, not decisions-to-have-a-meeting. A complex problem's probe is a specific experiment with a metric and a date. A complicated problem's analysis has an analyst and a timeline. A confusion problem's breakdown has a person and a return date.

What to watch for:

- Complex problems collapsed into complicated actions. *"For this complex problem, the action is we'll analyse it by Friday."* That's a complicated-domain action. Redirect: *"Give me the probe. What's a small, cheap experiment we could run?"*
- Chaotic heroics. The action for a chaotic problem becomes an inspirational speech about leadership. Redirect to mechanics: *"What are the three things we do in the next hour to stabilise?"*
- Unowned actions. Every action gets an owner or it's not an action. *"Who?"* until you have a name.

Worked probes for the complex domain, because *"design a safe-to-fail experiment"* is exactly where first-time facilitators freeze:

- *Subscriber churn is up and we don't know why.* Probe: ship a one-week pause-reason micro-survey to 50 churned subscribers. Metric: response rate plus dominant theme. Decision rule: if a single theme accounts for >40%, treat as complicated and analyse; if responses are scattered, run a second probe with longer-form interviews on a subset.
- *Team morale has been sliding.* Probe: introduce one structural change (e.g. half the standups become async, or one Friday is no-meetings) for two sprints. Metric: at the second retro after the change, has morale shifted? Decision rule: if better, amplify; if worse, dampen and try a different lever; if no change, the cause isn't where we thought.
- *We keep having near-misses on Friday pack-out.* Probe: rotate one extra person onto Friday pack-out for the next four Fridays. Metric: near-miss count plus their qualitative observations. Decision rule: if pattern emerges, you've moved into the complicated domain; if not, run a different probe (different shift pattern, different supplier mix).

The shape: small, cheap, time-boxed, with a metric and a decision rule. *"We'll think about it"* is not a probe.

Phase 6. Wrap up and commit (5 min)

Photograph the canvas. Read the action paradigms and owners aloud. End on commitments.

> *"We'll revisit this canvas in two weeks. The complex problems' probes should have produced data by then; we'll look at what emerged. The confusion problems should have been broken down; we'll see what the pieces look like."*

Worked example. See Cynefin: Not Everything Needs a Workshop for the Greenbox team's first Cynefin session, including the moment they realise the churn problem they've been treating as complicated ("we just need to analyse the data") is actually complex, and the different shape of the response that unlocks.

What Can Go Wrong

Everything is complex. Common when the team has been burned by complicated-domain tools that failed. *Recovery:* Pick one placement and pull at it. *"If a very good engineer analysed this for a week, would they find the root cause?"* If yes, it's complicated. If no, you've confirmed complex honestly. *Stop if:* The team won't place anything outside complex. They're using the framework to justify a mood. End and return when the mood has moved.

Domain-shopping. Someone keeps placing problems wherever justifies the response they've already decided on. *Recovery:* Separate the placement from the action. *"Let's place it first against the shape of the cause. We'll pick the action afterwards."* *Stop if:* The shopping persists. Name it: *"Are we placing this honestly or are we placing it where our preferred action is?"*

Cynefin as jargon. The team throws around *"sense-categorise-respond"* without converting it to action. *Recovery:* Ban the jargon for the next ten minutes. *"What would we actually do Monday?"* Use plain language until the action is concrete. *Stop if:* The team can't produce concrete actions. They don't own the framework yet; brief harder next time.

Chaotic heroics. Chaotic-domain problems become occasions for leadership grandstanding rather than stabilisation. *Recovery: "What stops the immediate bleeding? Who's going to do that in the next hour?"* Mechanics, not speeches. *Stop if:* The team treats chaos as a career opportunity. Have a private word; it will damage the culture if unchecked.

Refusal to use confusion. The team places everything definitively, usually as complicated. *Recovery: "Name one thing you genuinely don't know yet."* Whatever they name goes in confusion. Honesty modelled once often opens the rest. *Stop if:* The room won't admit uncertainty. This is a trust problem, not a framework problem.

The philosophical rabbit hole. Someone wants to debate Snowden's underlying theory. *Recovery: "Take it for coffee after the session. Right now we're using the working tool, not reviewing its foundations."* *Stop if:* The debate won't end. The session is a different conversation from the philosophy; reschedule.

Next Steps

The session ends; the work begins.

Same day, the facilitator:

- Photographs the canvas in panoramic, and each domain in close-up.
- Transcribes the list: each problem, its domain, its action paradigm, its owner, its date.
- Sends a one-page summary message: here are the problems, here are the domains, here are the actions we've committed to, here's when we'll look at the canvas again.

This fortnight, the delivery lead:

This is where the pattern earns its cost.

- Lets the complex-domain probes actually run. The temptation is to intervene and try to analyse them into complicated-domain answers. Resist. The probe's job is to produce data; the team's job is to watch.

- Pulls the confusion-domain problems back to the group. After the breakdown, the pieces return. Some will move into clear or complicated; some will stay complex; some will still be in confusion and need further breakdown.
- Revisits the canvas in two weeks. Some placements will have moved: a complex problem whose probe revealed a root cause may now be complicated; a complicated problem whose analysis didn't land may be complex. Update the map.
- Watches for domain creep. Problems don't sit still. A previously-clear problem may have grown into complicated because the system changed; an old complex problem may have resolved into complicated through emergent understanding. The canvas is a living artefact.

Ongoing, the team:

- Pins the canvas somewhere visible. When a new problem arrives, place it before picking an action.
- When a retro produces an action that doesn't stick, asks: *"was this in the correct domain?"* A misfitted action is a signal to re-place the problem.
- Uses domain placement in one-to-ones. *"How are you treating this problem, complicated or complex?"* is a faster diagnostic than *"how's that going?"*

Variants

Standard team session (default). 4-6 people, seventy-five minutes, 8-20 problems. Output: a populated canvas, action paradigms, owners and dates, a revisit in two weeks. This is what most teams need, and the rest of this post describes it.

Single-problem deep-dive. One contested problem, thirty minutes, the smallest group who can meaningfully contest the domain. Useful when a recurring action keeps not sticking and the team suspects a domain misfit. Skip the dump phase; the problem is the agenda.

Strategy-level pass. Senior leadership, ninety minutes, problems pitched at the level of *"we're losing in this market"* rather than *"the build is flaky."* The argument phase tends to be longer; the action paradigms are programme-shaped rather than ticket-shaped. Keep the framework honest; senior rooms will reach for complicated-domain analysis even on visibly complex problems.

Remote. Miro or Mural with the canvas drawn out, video call for the conversation. The dump phase is faster (everyone types in parallel); the placement phase is slower (the physical "carry the sticky over there" gesture is a useful think-aloud that disappears online). Use one shared cursor for placement, prompted by the team, to keep the layout legible.

Cynefin-on-the-wall. A permanent canvas pinned in the team room, used continuously rather than as a workshop. New problems get placed as they arrive; placements get re-argued at the regular team cadence. The workshop becomes a quarterly tune-up rather than a primary tool.

About this playbook

This playbook is part of *The Workshop*, a reference series of facilitator playbooks published at barkingiguana.com. The canonical, up-to-date version lives at barkingiguana.com/writing/the-workshop-cynefin-sensemaking/.

These posts are LLM-aided. Backbone, original writing, and structure by Craig. Research and editing by Craig + LLM. Proof-reading by Craig.