THE WORKSHOP

Which Workshop When

Sixteen techniques for discovery, design, and delivery. A flowchart and a categorised index for choosing the one that fits the problem in front of you. Pick the smallest workshop that solves the actual problem, not the biggest one your team knows how to run.

2026-11-27
barkingiguana.com/writing/the-workshop-which-workshop-when/

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

Contents

The hardest part of running discovery and design workshops isn't running them. It's choosing the right one. Most teams either stick to a single technique they know and apply it to every problem, or run every technique on every story until the team is sick of stickies. This post is the one-page index for choosing well: a flowchart for the most common starting points, then a categorised table of every technique with a one-line "use this when…" trigger.

About this index

Sixteen techniques. Each one solves a specific problem. None of them are useful when applied to the wrong problem. The point of this post isn't to advocate for the techniques individually, the linked posts do that. The point is to make the *choice* easy. Read the situation in front of you, find the matching row, run the named workshop. The next post you need is the one the link points to.

Intent

This is a *meta-workshop* in the Workshop series. Other posts in the series teach you how to *run* a single technique. This one teaches you how to *pick* one.

The goal is to remove the friction between "we have a problem" and "we're running the workshop that addresses it." If you're staring at a fuzzy situation and don't know which technique to reach for, this is the page that resolves the question.

When to use it

- A team meeting keeps not landing on a decision.
- The work feels disorganised but you can't articulate why.
- You're considering "let's run a workshop" but you're not sure which one.
- Something specific has gone wrong (high churn, monolith pain, dependency surprises) and you want to know which technique addresses it.
- You're reviewing your team's practices and want to know which workshops you're missing.

Participants

You. This is a one-person decision tool, then a workshop-selection conversation with whoever's facilitating.

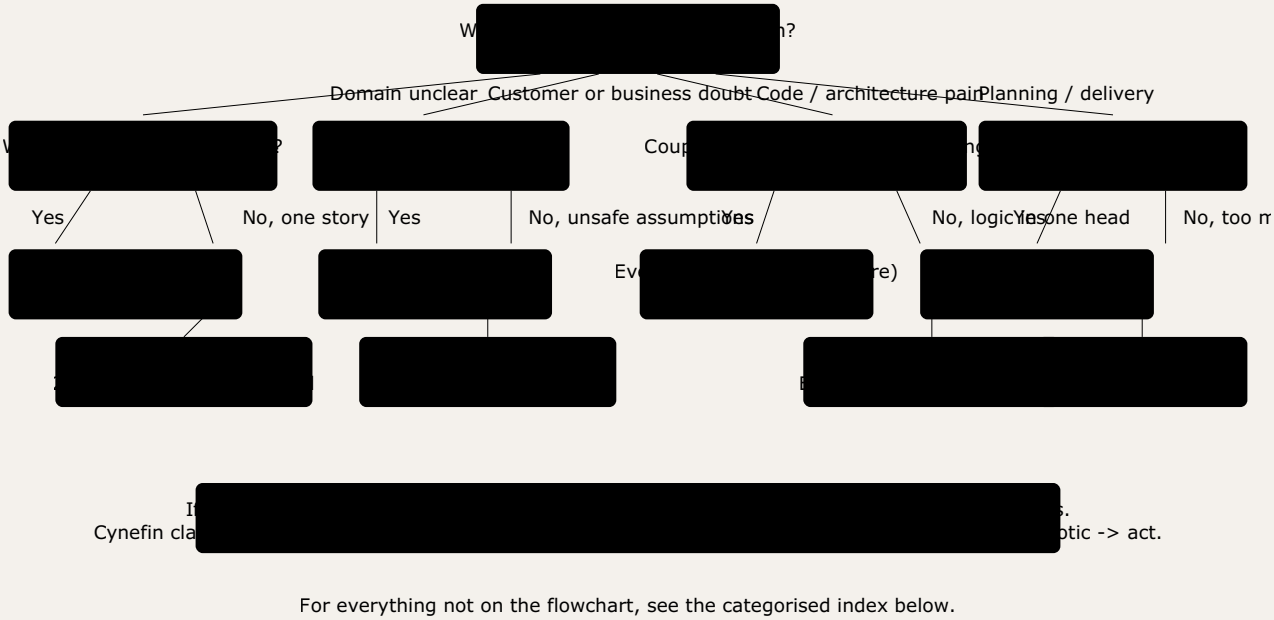
Materials and timing

Phase	Duration	Materials	Key question
Read the situation	1 min	,	“What’s the most pressing question?”
Walk the flowchart	2 min	Flowchart below	“Which gate does my situation match?”
Confirm with the index	1 min	Tables below	“Is this the right workshop or is there a closer fit?”
Post the choice	1 min	Team channel	“What am I running and why?”
Total	~5 min		

The point of a meta-workshop is that running it shouldn’t take longer than the workshop you choose. Five minutes is the upper bound. If you’re spending more time than that picking, the table isn’t pointing clearly enough, write to me and I’ll fix it.

The flowchart

Start with the most pressing question, follow the gates, land on a technique. The flowchart covers the most common entry points. The full table below covers everything.



For everything not on the flowchart, see the categorised index below.

The flowchart covers the four most common starting points. For everything else, use the table.

Discovery and domain understanding

Use this when…	Workshop	What you get
Your team doesn't share a mental model of how the business works. Assumptions are invisible.	Event Storming (Run it)	A shared timeline of domain events on a wall. Hotspots and unknowns made visible. Everyone sees the same picture.
Many possible features, no way to prioritise. Where does value flow and where are the bottlenecks?	Value Stream Mapping	A map showing which steps create value, which are waste, and where to focus first.
User stories are vague, "subscribe to a box" could mean five different things to five people.	Example Mapping (Run it)	Concrete, testable examples for each business rule, plus explicit unknowns to resolve before building.
You have concrete examples but need to turn them into working, tested code.	BDD / Gherkin	Executable specifications. Examples become automated tests. LLMs accelerate the code without losing rigour.
Features are shipping but the business metric isn't moving. No connection between work and goals.	Impact Mapping (Run it)	A map connecting deliverables to impacts to actors to a measurable goal. Reveals which features actually matter.
Long backlog, no sense of the full user journey. Hard to plan a coherent release.	User Story Mapping (Run it)	A timeline of the user's journey, sliced into releasable increments that make sense end to end.

Customer and business understanding

Use this when…	Workshop	What you get
Churn is high and nobody knows why customers leave, or why they stay.	Jobs to Be Done	The actual job customers hire your product for. Clarity on what's core to retention versus what's nice-to-have.
A key assumption turned out to be wrong. What else does the team believe that hasn't been validated?	Assumption Mapping (Run it)	A grid of assumptions by risk and evidence. Cheap experiments designed for the riskiest beliefs first.
The product works but the unit economics don't. Does the business model sustain itself at scale?	Business Model Canvas (Run it)	A one-page view of the nine business-model blocks. Dependencies and second-order effects visible.

Architecture and scaling

Use this when…	Workshop	What you get
Your codebase is a monolith. A simple feature touches dozens of files because everything is tangled.	Event Storming (Architecture) then DDD Modelling	Event Storming surfaces the seams; DDD Modelling sharpens them into bounded contexts with explicit language and loose coupling via events.
Critical business logic lives in one person's head. It can't be delegated, taught, or scaled.	Decision Tables (Run it)	A formal table of every condition combination and its correct outcome. Domain expertise made explicit and testable.
Decisions made months ago are forgotten. New developers guess at intent and guess wrong.	Architecture Decision Records	Short documents capturing what was decided, why, what alternatives were considered, what the consequences are.
Every capability looks worth building because LLMs make it cheap. But should you build, buy, or borrow?	Wardley Mapping (Run it)	A strategic map showing which capabilities differentiate you (build) and which are commodity (buy).

Team practices at scale

Use this when…	Workshop	What you get
One developer uses the LLM □ LLMA neural network trained to predict the next token in a sequence, large enough that it generalises to tasks it wasn't explicitly trained for.		
solo. The code arrives complete but nobody else understands it.	Ensemble Programming (Run it)	The whole team thinks while the LLM types. Problems caught in real time. Everyone understands the result.
Your team applies every discovery technique to every story. Workshop fatigue is setting in.	Cynefin	A classification (Clear / Complicated / Complex / Chaotic) that determines the right level of discovery for each piece of work.
The LLM writes code that works and passes tests but doesn't think about what could go wrong.	Threat Modelling (STRIDE) (Run it)	A systematic checklist of security threats at system boundaries, with mitigations planned before code ships.
An incident reached customers. Who's to blame? Wrong question.	Blameless Post-Mortems	A structural understanding of why the system allowed the failure, plus two or three concrete fixes with named owners.

Planning and delivery

Use this when…	Workshop	What you get
Discovery is done but there's no delivery rhythm. Work starts and finishes whenever.	Sprint Planning (Run it)	Two-week iterations with sprint goals connected to the Impact Map. Daily standups. Sprint reviews. Progress visibility.
Too many insights, not enough prioritisation. You know what's wrong but can't decide what to fix first.	Now / Next / Later	A prioritised plan with quarterly themes and measurable outcomes. A story the board can understand.
Multiple squads keep surprising each other with dependencies. No coordination between sprint plans.	Quarterly Planning Day	Aligned themes per squad, mapped dependencies, sprint coordination between teams.
Great weekly habits but no shared strategy connecting them. Squads locally excellent, globally confused.	The Planning Onion	Vision/Year/Quarter/Sprint/Day layers connected. Every sprint traces back to a strategic goal.
Sprint habits are strong but retros feel stale or don't produce change.	Retrospectives at Every Scale	Structured reflection at sprint, project, and org scale. Actions that actually get done.

Where to start if you adopt one thing

If you only adopt one technique, make it Example Mapping.

It's the shortest, twenty-five minutes per session. The most structured, four colours of card, one user story, a timer. The most immediately useful, you walk out with concrete examples your developers can build from. The cost of trying it once is minimal. The information about whether your team's discovery practice needs more is immediate.

The other techniques scale up from there. Map a domain when you don't share one. Map customer jobs when retention is unclear. Map architecture when the code's structure no longer matches the problem. Map dependencies when squads keep surprising each other. Each new map answers a different question.

What can go wrong

A few patterns that make workshop selection itself go sideways.

Picking the biggest workshop the team knows. Event Storming a single-team feature when Example Mapping would have done is a way to feel rigorous and waste two hours. Match the workshop to the *size* of the question.

Running the same technique on every problem. Every team has a favourite. Hammer-and-nail applies. If your last six workshops were all Example Mapping, you're missing the categories of question that Example Mapping doesn't answer.

Workshopping a problem a conversation can solve. If three people agree in five minutes at a desk, that's the workshop. Don't schedule a session for what a chat resolves. Cynefin gives you the language for this: *Clear* doesn't need a workshop.

Skipping the workshop because the answer feels obvious. The opposite failure. Every founder thinks the customer's job is obvious. Most are wrong. The workshop exists to surface the assumption you're confident about.

Treating workshops as ritual. A workshop run on autopilot, with no clear question to answer, produces tired stickies and no decisions. Every workshop in this index has a *trigger condition* in the "Use this when" column. If your situation doesn't match, run something else (or nothing).

Outputs

This is a meta-workshop, so the output is the *next workshop you run.* Pick a row from a table. Click the "Run it" link. Use that playbook.

Optionally, paste the chosen technique link into the team channel with a one-sentence rationale: *"Running Example Mapping on the renewal flow tomorrow morning, too many edge cases for the current ticket."* That note is its own piece of communication infrastructure: it tells the team what's about to happen and why.

Where to go next

The Workshop series itself, in technique order:

- Event Storming a Domain, big-picture, whole-business
- Event Storming a Process, the default session, three hours
- Example Mapping, twenty-five minutes per story
- Impact Mapping, connect work to goals
- User Story Mapping, the whole journey
- Jobs to Be Done, why customers hire you
- Assumption Mapping, test what you believe
- Business Model Canvas, nine blocks on one page
- Decision Tables, formal logic
- Sprint Planning, delivery rhythm
- Blameless Post-Mortems, structural learning from failure

For the planning context that holds it all together, see The Planning Onion.

For worked examples of every technique, see The Greenbox Story.

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-which-workshop-when/.

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