THE WORKSHOP

Wardley Mapping

A pattern for laying the components of a value chain (the sequence of capabilities a user needs, top visible to the user, bottom foundational) against an evolution axis, so that build-buy-borrow arguments stop being about taste and start being about where each component actually sits.

2026-10-23
barkingiguana.com/writing/the-workshop-wardley-mapping/

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

Contents

Build it, buy it, or borrow it? Wardley Mapping shows which capabilities differentiate you and which are commodity, so you stop building things AWS already sells for the price of a coffee. Worked example: Build, Buy, or Borrow?.

Wardley Mapping

Wardley Mapping plots the components that serve a user need against two axes, visibility to the user (vertical) and stage of evolution from novel to commodity (horizontal), so build/buy/borrow decisions are made from a picture instead of an argument. Named after Simon Wardley, who developed it at Fotango in the early 2000s and has been giving it away free ever since. Frequently confused with architecture diagrams (which describe structure) and value stream maps (which measure time through a process); a Wardley Map plots strategic position.

At a glance

- *Who, for how long:* a facilitator who doesn't place components, a product or strategy person, two or three tech leads, and someone with market exposure (advisor, consultant, anyone who reads release notes). Four to six people, two hours.
- *What you walk out with:* 15–25 components on the map with movement arrows on each, and three to six concrete build / buy / borrow decisions tagged with owners and dates.
- *When to reach for it:* a build/buy/borrow argument going in circles, or strategy planning where you need to separate the parts of the stack that differentiate you from the parts that are plumbing. Not for sprint planning, feature prioritisation, or mapping a domain you don't yet understand (do Event Storming first).

What's It For

A team is about to spend a quarter building a custom queue. Someone mentions SQS. The argument that follows is the familiar one: "we need the flexibility," "we can't be locked in," "our needs are unique." The argument is really about taste, because nobody in the room has a shared language for saying *this is a commodity and building it is waste,* or *this is genuinely novel and buying it would leave us with someone else's compromises.* Without that language the loudest voice wins, and the loudest voice is often the one that wants to build.

The same argument plays out in SRE conversations weekly. *"Should we run our own Kafka or use a managed one?" "Should we build a feature-flag system or use LaunchDarkly?" "Should we self-host observability or pay for Datadog?"* These are strategic questions dressed up as technical ones, and the team that answers them by taste ends up in the worst of both worlds: running commodities they shouldn't run, and paying vendors for things they could have built into a moat.

A Wardley Map replaces the argument with a picture. You list the components needed to serve a user need. You position each one by how visible it is (a user-facing login page at the top, a database at the bottom). You then slide each one horizontally to reflect its stage of evolution: Genesis for novel and uncertain, Custom-built for bespoke-because-nothing-exists, Product for off-the-shelf, Commodity/

utility for pay-per-use plumbing. Suddenly the conversation is different. *"Why are we building this? It's in the Commodity stage, look." "Because the off-the-shelf options don't cover the domain logic that makes us distinct, look."* The map doesn't end the debate; it changes what the debate is about.

Reach for it when:

- You need to make a build/buy/borrow decision and the arguments are going in circles
- You're planning strategy and need to distinguish the parts of the stack that differentiate you from the parts that are plumbing
- An SRE team is deciding whether to self-host or buy a managed service
- You want to anticipate where the market is heading, not just where it is today
- You're onboarding a new technical leader who needs the strategic landscape on one page, what you're building, what you're buying, what's commoditising, where the moats are
- You suspect your team is building commodities that should have been bought years ago

What It's Not For

Skip it when:

- You need to plan a sprint or prioritise features; this is strategy, not delivery
- You're in the first few weeks of a project and don't yet know the domain; map the domain first with Event Storming, then map the strategy
- Nobody in the room understands the market well enough to place things on the evolution axis

Trade-offs to weigh before you commit.

Benefits:

- Build/buy/borrow arguments move from taste to evidence
- Commodity components you've been building become visible, and are usually cheaper to stop building than to keep
- Differentiating components become visible too, and get the investment they deserve
- The team gets a shared language for talking about strategic position
- New technical leaders can look at one picture and understand the strategic landscape

Costs:

- A focused 2 hours from 4-6 people, including scarce senior time
- A map is only as good as the market knowledge in the room; a team that hasn't looked outside in a year will misplace everything and believe the misplacement
- The map will surface decisions that somebody is emotionally invested in and doesn't want surfaced
- First maps are rough, and a rough map can feel like a waste if the team expects polish

Failure modes:

- The map turns into an architecture diagram and the strategic conversation never happens
- The team places everything as Custom-built because they haven't looked at the market
- The map surfaces decisions but no-one commits to action, and the map ages on a wall

- The wrong people are in the room: all technical, or all executive, or all one team

Stop signals:

- An hour in and the value chain still isn't on the wall
- The evolution axis is being used to describe the team's codebase, not the market
- Strategic decisions keep getting deferred to a person who isn't in the room

The real cost of a bad Wardley Map isn't the two hours spent drawing it; it's the quarter you spend afterwards building something you should have bought, or paying a vendor for something that would have been a moat. The map's whole point is to make those two outcomes visible before they happen.

Definitions & Background

The two axes.

- Y-axis, vertical: visibility to the user. Top is "the user directly interacts with this." Bottom is "the user has no idea this exists."
- X-axis, horizontal: evolution. Four zones, left to right: Genesis (novel, uncertain, rapidly changing), Custom-built (understood but bespoke), Product (off-the-shelf, multiple vendors), Commodity/utility (standardised, pay-per-use, boring).

The evolution stages in detail. Evolution is driven by ubiquity (how widespread something is) plus certainty (how well-understood it is). Things move right as both increase. Wardley's stage characteristics:

Stage	Ubiquity	Certainty	Knowledge	Differential
Genesis	Rare	Uncertain	Changing	Different from everything else
Custom-built	Uncommon	Learning	Diverging	Bespoke to each user
Product (+ rental, where someone runs it for you)	Increasing	Converging	Good practice	Multiple competing implementations
Commodity (+ utility, where it's metered like electricity)	Ubiquitous	Certain	Standard	Essential, undifferentiated, pay-per-use

Climate. The map is a snapshot, but the things on it are moving. Competitive pressure pushes components rightward over time: today's Custom-built becomes tomorrow's Product becomes the day-after's Commodity. Wardley calls this set of pressures *climate*, the weather acting on the map, not a decision the team gets to opt out of. Whether you redraw the map or not, the components are drifting.

Inertia. The forces that hold a component in place against the climate: past success, sunk cost, political capital, skills the team has invested in, vendor contracts, fear of switching. Inertia is real and worth naming on the map, but it explains why a component hasn't moved, not whether it should. *"We can't migrate because the on-call rotation is built around it"* is inertia; it's a cost to overcome, not a reason to stay.

Levels. Wardley Mapping runs at three zooms; the level changes the scope and the strategic horizon.

Level	Scope	Duration	Output
Component Level *(zoom in)*	One specific build/buy decision and its dependencies	60-90 min	Decision + justification
Standard *(default)*	One user need, its full value chain	2 hours	Strategic map, 3-6 build/buy/borrow decisions, watch-list
Portfolio Level *(zoom out)*	Multiple user needs and the shared components between them	Half day	Portfolio view, cross-team dependencies surfaced

The Standard level is where most teams should start. A single user need, *"a subscriber receives a weekly box of seasonal produce"*, and the full chain of components beneath it. Zoom in to Component Level only when there's a specific decision that's stuck and you don't need the full picture. Zoom out to Portfolio Level only after you've mapped the individual user needs and want to see where they share components.

Inputs

- A user need framed as a sentence. *"A subscriber receives a weekly box of fresh, seasonal produce delivered to their door."* The map is built to serve this need; everything on it has to connect back to it.
- A large surface, whiteboard, butcher paper, or a digital canvas. The map grows wide as well as tall, so don't pick a small one.
- Sticky notes in at least two colours. One colour for components, a different colour for strategic decisions (BUY, BUILD, WATCH, MIGRATE) in Phase 5. Arrows for movement annotation can be drawn directly or use a third colour.
- Two hours, uninterrupted, with the right people in the room (see *Who's Needed*).

Bring nothing else except what you know about the market. The map is generative, the value is in placing and arguing, not in preparation.

Outputs

What lands on the wall at the end:

- A populated map with 15-25 components placed by visibility and evolution, connected by dependency lines.
- Movement annotations, arrows showing where each component is heading and how fast.
- Marked inertia, bars across components naming what's holding them in place (skills, vendor contract, sunk cost).
- 3-6 strategic decisions as different-coloured stickies on the map: BUY, BUILD, WATCH, MIGRATE, each tied to a specific component.
- A watch-list of components that aren't decided today but need a review date in the calendar.

Photograph the map from multiple angles and in good light before anyone walks away.

These outputs feed straight into:

- Event Storming. Event Storming describes how things happen in one process; Wardley Mapping describes the strategic shape of the components that do the happening. Run Event Storming first to understand the domain; run Wardley Mapping to decide which parts of it are yours to build.
- Business Model Canvas, the Business Model Canvas frames *what* you're doing and for whom. Wardley Mapping frames *how* you're doing it and which bits matter. Canvas first, map second: the map is most useful once you know which user need to map.
- Assumption Mapping, a Wardley Map surfaces strategic risks. Assumption Mapping turns those risks into testable assumptions. A BUY decision is really an assumption about whether the vendor will meet your needs, that assumption is worth testing deliberately.
- Impact Mapping. Impact Mapping starts with the outcome you want and works backwards. Wardley Mapping starts with the user need and works across. Run them together when you're planning a quarter: Impact for *why*, Wardley for *how*.

Who's Needed

Facilitator. Does not place components. Their job is to draw the axes, keep the value chain moving downward, force the conversation about evolution to actually happen, and stop the map from turning into an architecture diagram.

A product or strategy person who understands the user need at the top of the map and the competitive landscape around it. Without this person, the map floats free of why any of the components exist.

Technical leads who know the current architecture and the available alternatives. At least one should be strong on *what else exists in the market.* A team that only knows its own stack will place everything as Custom-built because they have never looked at the other columns.

Operations or SRE people, essential when the map touches infrastructure. They know which "custom" components are actually three scripts and a cron job, which managed services have matured, and which vendors will page you at 3am on a holiday. When the map is about the platform itself rather than the product that sits on it, they lead.

Someone with market exposure, a technical advisor, a consultant, someone who reads release notes. Evolution is a market question, not a team question, and a team that hasn't looked outside in a year will misplace everything.

Group size: 4-6. Below four and you don't have enough angles on evolution; above six and the placement debates collapse under their own weight.

Who to leave out:

- People who are emotionally attached to a specific component. The person who built the in-house queue three years ago will find it hard to place it in Commodity even when it obviously belongs there. If they must be in the room, name the dynamic up front: *"We're mapping the market, not judging anyone's past decisions."*

- Executives who will override the map with a pre-made decision. If the CTO has already decided what to build, running the workshop is theatre. Either get the decision on the table first or don't run the session.
- Spectators. Wardley Mapping is participatory. Passive observers distort the dynamic without contributing.

How To Run It

Phase	Duration	Materials	Key question
Frame the user need, draw the axes	10 min	Large surface	"Who is the user and what do they need?"
Build the value chain	25 min	Sticky notes, one colour	"What do we need to provide that?"
Assess evolution	30 min	Same notes, sliding horizontally	"Is this novel, custom, product, or commodity?"
Annotate movement	15 min	Arrows, different colour	"Where is this heading? How fast?"
Break	10 min	,	,
Strategic decisions	20 min	Different-coloured sticky notes	"Build, buy, or borrow? Where are the risks?"
Wrap-up, owners, next steps	10 min	,	"What do we do this week?"
Total	2 hours		

The four working phases are 90 minutes. The remaining time is the intro, the break, and the wrap-up. Don't skip the break, phase 3, evolution, is the most intellectually tiring, and people need a pause before strategic decisions.

The session has two distinct modes. The first two phases (value chain, evolution) are group placement: everyone contributes, the facilitator mediates disagreements, and the map grows organically. The second two phases (movement, decisions) are directed argument: the facilitator asks each question of the group and pushes for a concrete answer.

The key rhythm is place vertically first, then slide horizontally. People want to debate evolution the moment they see a component. Resist. Get the whole value chain on the wall before touching the evolution axis. Once the chain is there, the evolution conversation has context.

Phase 1. Frame the user need, draw the axes (10 min)

Before anyone places anything, draw the map's skeleton:

- Y-axis, vertical: visibility to the user. Top is "the user directly interacts with this." Bottom is "the user has no idea this exists."

- X-axis, horizontal: evolution. Four zones, left to right: Genesis (novel, uncertain, rapidly changing), Custom-built (understood but bespoke), Product (off-the-shelf, multiple vendors), Commodity/utility (standardised, pay-per-use, boring).

Write the user need at the very top of the y-axis, as a sentence:

> *"A subscriber receives a weekly box of fresh, seasonal produce delivered to their door."*

What to say:

> *"Everything we put on this map has to serve this user need. If it doesn't, it doesn't belong here. We're going to fill in the value chain first, what does the user see at the top, what does that depend on, what does that depend on, and once the whole chain is on the wall, we're going to slide each component sideways to reflect where it sits in the market."*

What to watch for:

- A user need that's really a feature. *"The checkout page loads quickly"* is a feature. *"A subscriber chooses and pays for a subscription"* is a user need. Push up a level if needed.
- Multiple user needs smuggled in as one. *"Subscribers order and receive produce and manage their subscription and contact support"* is four user needs. Pick one for this map; save the others for next time.
- Skipping straight to components. Someone wants to write "Postgres" before the user need is framed. *"Hold that, we'll get there. First, who is this for and what do they need?"*

Phase 2. Build the value chain (25 min)

Starting from the user need at the top, work downward by asking *"what do we need to provide that?"*

One layer at a time. For the subscription example above, the first layer might be: a subscription management system, a box curation process, a delivery service, a payment system. For each of those, ask again: *what does this depend on?*

Write each component on a sticky note and place it on the map, positioned vertically only by how visible it is to the user. Don't think about evolution yet. A login page is near the top. A database is near the bottom. A delivery experience is somewhere in the middle, the user sees the result but not the logistics.

Draw lines between components that depend on each other. The chain should read like a dependency tree, with the user need at the top and the lowest-level commodities at the bottom.

What to say:

> *"What do we need to provide this? Give me the first layer."*

> *"And what does that depend on? Keep going until you hit something that's obviously a commodity, or obviously a single thing."*

> *"Farm relationships count. Seasonal knowledge counts. Customer support counts. A value chain includes people and processes, not just software."*

What to watch for:

- Jumping to evolution too early. Someone wants to say *"well, that's a commodity"* as soon as a component goes up. Redirect: *"We'll slide it sideways in the next phase. Right now, just place it vertically."*
- The map becoming an architecture diagram. If every component on the wall is a software box, ask explicitly: *"Where are the human components? The knowledge? The supplier relationships? Those belong too."*
- Stopping too high. If the lowest component on the wall is "the web application," you haven't decomposed enough. *"What does the web app need? Hosting? A runtime? A database? A CDN? Keep going."*
- Decomposing infinitely. You don't need to get down to individual CPU instructions. Stop when a component is either obviously a commodity or obviously a single thing the team owns.

By the end of phase 2 the wall should hold 15-25 components connected by dependency lines, positioned vertically but all clustered on the left. That's fine, the next phase spreads them.

Phase 3. Assess evolution (30 min)

Now slide each component horizontally. This is where the strategic conversations happen.

Evolution is driven by ubiquity (how widespread something is) plus certainty (how well-understood it is). Things move right as both increase. For each component, ask the group:

- Is this novel and uncertain? Genesis.
- Is it understood but bespoke because nothing good exists? Custom-built.
- Are there multiple competing off-the-shelf options? Product.
- Is it standardised, interchangeable, pay-per-use plumbing? Commodity.

Work through the components one by one. Let the team discuss each one before moving it. The debates are the point.

What to say:

> *"If we were starting today with no code at all, would we build this, or would we buy it? If we'd buy it, where would we buy it from? Is there one vendor or ten?"*

> *"Can you name three alternatives that would do this job? If yes, we're at least in Product. If no, we're probably Custom-built."*

> *"This isn't about what we have. It's about what the market has. The component's position reflects where the market is, not where our code is."*

What to watch for:

- Emotional attachment. Someone places a component in Custom-built because they built it, even though three SaaS vendors do the same thing. Use the fresh-start question: *"If we were starting today, would we build this?"* If the answer is no, the position is Commodity.
- Confusing "we built it" with "it needs to be built." These are different. The map reflects the second.

- Useful disagreement. Two people disagree on whether something is late Custom-built or early Product. Excellent, this means one person knows about vendors the other doesn't. Let it play out; share the information.
- Everything clustering in the middle. If every component ends up in Product, the team hasn't thought hard enough about what's genuinely Commodity (cloud compute, email, payments, DNS) or what's genuinely Custom-built (that bespoke algorithm nobody else has). Push both ways.
- The "we're special" claim. *"Our needs are unique, this has to be custom."* Unique in what way? Which specific requirement rules out every off-the-shelf option? If no-one can name it, it's not as unique as it feels.

Phase 3a. Climate and inertia (10 min)

Components don't sit still. The Red Queen pressure (see *Definitions & Background*) pushes everything rightward over time, whether the team likes it or not. Today's Custom-built becomes tomorrow's Product becomes the day-after's Commodity.

Walk the wall and mark inertia where you see it, a small bar across a component, labelled with the kind. *"Skills inertia, the team built this and would lose them." "Vendor inertia, five-year contract." "Sunk-cost inertia, two years of investment."*

Naming inertia changes the conversation. The map is no longer "what should we do?", it's "what's actually possible to move, given what's holding us still?" The migrate decisions in Phase 5 then either accept the inertia (and the climate keeps pushing the cost up) or pay to overcome it explicitly.

Phase 4. Annotate movement (15 min)

The map so far is a snapshot. Now make it a forecast. For each component, ask: *"Is this moving right? How fast?"*

Draw arrows to show direction and speed of evolution:

- A long arrow from Custom-built toward Product: *"this is commoditising fast, if we're building it, we should stop."*
- A short arrow: *"slow drift, review in a year."*
- No arrow: *"stable for the foreseeable future."*
- An arrow from Genesis toward Custom-built: *"we're figuring this out; it's not ready to buy, but it will be."*

What to say:

> *"When did we last check what's available for this? Has the market changed in the last eighteen months?"*

> *"Which of these components will look different in two years? What will be the first to commoditise?"*

What to watch for:

- Ignoring market movement. The team may not have noticed that delivery logistics software has moved from Product to Commodity in the last two years. Prompt explicitly: *"When did we last check?"*

- Wishful arrows. Someone draws an arrow because they *want* their custom component to become commodity. Arrows reflect the market, not the wish.
- No arrows at all. Nothing is moving? Unlikely. *"Name one component that will look different in two years. Start there."*

Phase 5. Strategic decisions (20 min)

Now use the map. Walk component by component and ask:

- Build, buy, or borrow? Components on the left that are visible and differentiating, build. Components on the right that are invisible, buy or use a service. Middle components need a specific evaluation of what's available.
- Where are the risks? Custom-built components with fast rightward arrows are expensive to run while the market eats them. Single-supplier dependencies are fragility. Flag them.
- Where are the opportunities? Genesis components that only you are exploring might be competitive advantages. Components where you have expertise the market doesn't yet might be product opportunities.

Mark decisions directly on the map with a different-coloured sticky: BUY, BUILD, WATCH, MIGRATE.

What to say:

> *"For this component, what do we do this week? Next month? This quarter? I need a concrete answer, not 'we should consider it.'"*

> *"If we do nothing, what happens in eighteen months? Is that an acceptable outcome?"*

> *"We just said this is a commodity. We're also currently building it. What's the action?"*

What to watch for:

- Sunk-cost arguments. *"We've already built the payment system, so we should keep using it."* The map doesn't care what you've already spent. *"If we were starting fresh, would we build this?"*
- Build-everything bias. Developers usually prefer building. The map is a corrective.
- Buy-everything bias. The opposite problem. Some components genuinely need to be custom; if the thing that differentiates you is in the box curation algorithm, you should not outsource it.
- Analysis without decisions. *"We should think about this more"* is not a decision. Push for concrete next steps with owners and dates.

Worked example

See Wardley Mapping: Build, Buy, or Borrow?, the Greenbox team plotting their stack against evolution and finding that three things they were building belong in Commodity, and one thing they were about to buy is actually their biggest differentiator. The moment the map changes the decision is the moment the pattern earns its cost.

What Can Go Wrong

The architecture diagram. The map has turned into a microservices diagram with databases and queues and no humans or processes anywhere. *Recovery: "Where are the human components? The knowledge? The supplier relationships? A Wardley Map includes everything needed to serve the user need."* Add them explicitly. *Stop if:* The team can't think about non-software components at all. You have a technical architecture group, not a strategy group, rebook with product people present.

The evolution argument. Two people spend fifteen minutes debating whether something is late Custom-built or early Product. *Recovery:* Use the three-vendors heuristic. *"Can you name three competing products you could buy today? Yes? Then it's at least Product. Moving on."* *Stop if:* Every component produces the same debate. The team doesn't have enough market knowledge, reconvene with someone who does.

The missing user. The map has lots of components but no visible connection from any of them to the user need at the top. *Recovery:* Walk the chain from top to bottom out loud. *"Can I trace a path from the user need to this component? If not, why is it on the map?"* *Stop if:* Half the components are disconnected. The scope is wrong, you're mapping the org chart or the codebase, not a user need.

The too-detailed map. Forty components, nobody can see the strategic picture. *Recovery: "Which ten components matter most for the decisions we need to make? Let's focus there. Move the rest to a parking area."* *Stop if:* The team can't agree on which ten matter. The user need is too broad, split it into multiple maps.

The first-timer confusion. People are struggling with the axes and placements feel arbitrary. *Recovery:* Pause. Walk one component through end to end: *"User authentication. Visible? Yes, mostly, it's the login screen. Evolution? Commodity. Auth0, Firebase, Cognito. So it goes here: upper-middle on visibility, far right on evolution. Which means: we should buy it."* Then let the group do the next one. *Stop if:* Nobody can place a component even after a worked example. The team isn't ready for this workshop, they need a primer first.

The pre-decided decision. Halfway through, it becomes clear that someone senior has already decided what the map should say. *Recovery:* Name it. *"It sounds like there's a decision already in play. Let's put it on the map and see if the map agrees or disagrees."* *Stop if:* Naming it changes nothing and the session is theatre. End early; the decision belongs to the person who already made it, not to the room.

Next Steps

The session ends; the work begins.

Same day, the facilitator:

- High-resolution photographs of the map from multiple angles. Good lighting matters, a map you can't read is a map you won't use.
- A digital transcription. There are Wardley Mapping tools; any drawing tool works. Mermaid's Wardley syntax is a strong option if you want the map to live in the repo and render in GitHub, Notion, or Obsidian alongside the rest of your docs, anchors, components, evolution arrows, inertia, and the

build / buy / outsource decorators are all supported, so most of what's on the wall transcribes directly. The map is visual, so a spreadsheet is the wrong shape.

- A short summary to participants: *"Here's what the map showed. Here are the decisions. Here's who owns what."*

This week, the product owner:

- Turn each decision into an action with a name and a date. *"Evaluate Stripe by Friday week,"* not *"consider payment alternatives."* Without a specific owner and deadline, the map rots.
- For BUY decisions, begin the vendor evaluation. The map provides the justification; the evaluation provides the specifics. Match the seven to ten things your team actually needs against what each vendor offers. Include an exit plan: *what happens if we need to leave this vendor in two years?*
- For BUILD decisions, make the strategic importance visible in the backlog. A story for a differentiating component should be marked as such. The next person picking up the backlog should be able to see *why* it's being built.
- For WATCH decisions, book the review. Put a date on the calendar. *"Revisit this in Q3."* If you don't, it'll drift off the edge of the map.
- Walk the map past anyone who couldn't attend. Their reaction will surface decisions that need broader input.

Ongoing:

- Revisit the map every quarter. Evolution is slow but real; components drift right, and last year's custom-built advantage becomes this year's commodity.
- Use the map as a check on new proposals. When someone wants to build something, point at the map: *"Where does this sit on evolution? Are we building a commodity?"* Fifteen-second conversation instead of fifteen minutes.
- When the user need at the top of the map changes, the whole map needs revisiting. A new user need is a new map.

Variants

Standard (default). One user need, its full value chain, two hours, 4-6 people. Output: a strategic map with 3-6 build/buy/borrow decisions and a watch-list. This is what most teams need, and the rest of this post describes it.

Component Level *(zoom in)*. One specific build/buy decision and its dependencies, 60-90 minutes. Output: the decision plus its justification. Reach for this when there's a single stuck argument and you don't need the full picture, *"should we self-host this queue or use SQS?"* with the components either side of it on the chain, and nothing else.

Portfolio Level *(zoom out)*. Multiple user needs and the shared components between them, half a day. Output: a portfolio view that surfaces cross-team dependencies. Best run after the individual user needs have already been mapped at Standard level; the portfolio map is a synthesis, not a starting point.

Remote. A digital canvas (Miro, Mural, or a dedicated Wardley tool) with the axes pre-drawn and a video call for the conversation. Slightly slower than in-person, the rhythm of *"write a sticky, place a sticky"* is faster on a wall, but the structure transfers cleanly. Use one shared cursor: only the facilitator places components, prompted by the team, to keep the map legible. If the team wants the map to live in source control afterwards rather than in a canvas tool, Mermaid's Wardley syntax is a reasonable transcription target for the digital follow-up.

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-wardley-mapping/.

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