THE WORKSHOP

Decision Tables

A pattern for extracting a tangled business rule from a domain expert's head and laying it out as a conditions-and-outcomes grid, so the combinations nobody had thought of become visible before they become bugs.

2026-08-14
barkingiguana.com/writing/the-workshop-decision-tables/

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

Contents

"It depends" is how most business rules ship wrong; nobody enumerated what they depend on. A Decision Table forces the enumeration, and the expert's long pause is where the unshipped bugs live. Worked example: Making Maya's Brain Explicit.

Decision Tables

Decision Tables capture a complex business rule as a grid of conditions and outcomes, so that every combination has a defined answer and every answer has a reason. Decision tables in most of the literature, occasionally rules tables or condition-action tables. The technique goes back to the 1960s, formalised in structured analysis and business-rules engines, and it has survived every methodology fad since because the underlying problem never went away. Often confused with decision trees (which are hierarchical and good for explaining a flow) and with truth tables (which are a mathematical subset). A Decision Table is a business-rules instrument; the grid is where all the work happens.

At a glance

- *Who, for how long:* a facilitator who doesn't contribute rules, the actual domain expert (the person who currently makes this call), one or two developers, and a tester if you have one. Three to four people plus the facilitator, sixty to ninety minutes.
- *What you walk out with:* a completed grid with conditions, outcomes, and don't-cares; a named hit policy; a list of policy gaps the expert couldn't answer; and a set of test cases, one per column.
- *When to reach for it:* a business rule with multiple interacting conditions, "it depends" answers, and the same question producing slightly different answers each time it's asked. Not for inventing a rule from scratch (Decision Tables capture rules that exist), and not for a single `if` / `else` that's pattern-matching as complex.

What's It For

A domain expert says "it depends," and then talks for ten minutes, and at the end the developer writes an `if` statement that covers what they heard. Three weeks later a subscriber in an unusual situation gets the wrong box. The developer goes back to the expert. The expert explains again, slightly differently this time. The developer updates the code. Two weeks later, a different unusual subscriber: wrong box again.

The problem is not that the expert is wrong. The expert is correct every time they speak. The problem is that the rule lives in their head as a set of reflexes, and reflexes answer the case in front of them without ever enumerating the cases that aren't. The developer hears the instance and writes the instance. The combinations that nobody asked about never get written down, because nobody thought to ask.

A Decision Table forces the enumeration. You draw the grid, and suddenly there are eight columns, and the expert has an answer for six of them and a long pause for the other two. The long pauses are the bugs you haven't shipped yet. The grid is the sheet music for a song the expert has been playing from memory for years.

Reach for it when:

- A business rule has multiple conditions that interact: "if A and B then X, but if A and not B then Y, unless it's Wednesday"
- The domain expert describes the logic with "well, it depends on…" or "except when…"
- Developers have asked the same question multiple times and got slightly different answers
- Bug reports keep arriving for edge cases nobody anticipated
- You're writing a policy engine, an alerting rule, a pricing rule, a routing rule, or any other place where "what should happen here" is the hard question
- You're codifying an SRE runbook where the on-call engineer's decision depends on which alerts are firing together

What It's Not For

Skip it when:

- The rule is a single `if` / `else` and you're pattern-matching it to look complex
- Each condition has an independent effect; a bulleted list is fine
- You're trying to *invent* the rule from scratch. Decision Tables capture rules that exist; they don't design rules that don't

Trade-off costs and failure modes to keep in mind:

- A focused 90 minutes from 3-4 people, including the one person whose calendar is always full
- Emotional cost when the expert discovers they've been doing it wrong
- Possibility of surfacing a policy gap that somebody doesn't want surfaced
- A table needs maintenance; when the rule changes, the table needs to change too
- The expert redesigns the rule instead of describing it, and the table captures fiction
- The grid explodes past the point of legibility
- The scope was wrong and the session runs over without finishing
- The policy gaps are flagged and then sit unresolved, turning the table into an accusation

Stop signals. End the session if:

- Sixty minutes in and the first column isn't filled
- The expert is describing three different rules that can't be reconciled
- The developers have stopped asking questions because they're bored or lost

A Decision Table that surfaces a handful of combinations the team had never considered has already paid for itself; those combinations were bugs waiting to happen, and now they're rows in a grid. A domain expert saying *"oh, I've never thought about that combination"* is not a failure of the expert. It's the moment the session's value appears.

Definitions & Background

Don't-care cells. A cell whose value doesn't change the outcome, conventionally written as a dash (`-`). Don't-cares are how a six-condition rule that would otherwise need 64 columns collapses to a readable 12-20. They tell you "this dimension doesn't matter for this rule," and they're the reason a Decision Table stays legible as conditions multiply.

Hit policy. Once you allow don't-cares, two rules can match the same input. A subscriber who is "new" *and* has a "promo code" might match both the new-subscriber rule and the promo-active rule, and they might give different answers. The hit policy is the rule for picking which row applies when multiple match. The four standard options are:

- *Unique*: the columns must not overlap; if they do, it's a bug
- *First*: whichever rule matches first wins (top-to-bottom)
- *Priority*: each rule has a priority; highest wins
- *Any*: overlapping rules must agree on the outcome

Pick a hit policy and write it on the wall before you finish the table, or you'll discover the conflict in production.

DMN (Decision Model and Notation). The OMG standard for decision tables. DMN names the hit policy explicitly in the table header (with single-letter codes: U, F, P, A) and gives the grid a formal structure suitable for executable rules engines. You don't need DMN to run the workshop, but if your team is heading toward a rules engine, knowing the standard saves a translation step later.

Completeness vs consistency. Two distinct properties to test. *Completeness* asks: does every input combination have an answer? *Consistency* asks: when two rules can match the same input, do they agree (or does the hit policy resolve the conflict cleanly)? Phase 5 of the session walks both, in that order.

Levels. The level you pick changes the shape of the session. Start at Standard unless the rule is clearly a small one or clearly a huge one.

Level	Scope	Duration	Output
Rule Level *(zoom in)*	A single decision with 2-3 conditions	30-45 min	Small table, sometimes just a cleanup of an existing `if`
Standard *(default)*	One business decision with 4-6 interacting conditions	60-90 min	Complete table, policy gaps surfaced, test cases ready
Policy Level *(zoom out)*	A family of related decisions across a process	Half a day, possibly split	Multiple linked tables, often a policy review afterwards

The Standard level is where the pattern earns its keep. A single decision, "what box does this subscriber receive?" or "can this deployment proceed?", with enough conditions to make the interactions interesting. Rule Level is worth running when you suspect a small rule is hiding a bigger problem. Policy Level is what you reach for when the whole team is arguing about pricing, or when an SRE runbook has grown into a jungle.

Inputs

Bring nothing. The expert's head is the input, the whiteboard is the workspace.

- An erasable whiteboard big enough for ten-plus columns. You will rewrite the grid at least once as new conditions surface.
- One specific decision framed as a question: "What box does this subscriber receive?" not "Box assignment policy." If you can't write the decision as a question, the scope is wrong.
- The right people in the room (see *Who's Needed*). The single non-negotiable input is the actual domain expert: the person who currently makes this decision, not someone who knows a bit about it.
- A 60-90 minute slot with no interruptions for the Standard level.

Outputs

What lands on the wall at the end:

- A completed grid. Conditions in rows at the top, outcomes in rows at the bottom, columns enumerating the combinations the rule actually produces. Don't-care cells marked with a dash. Hit policy named on the header.
- A list of policy gaps. Cells where nobody in the room knew today's answer. Each gap is one of: (a) a policy decision the product owner can make, (b) a decision that needs authority from outside the team, (c) a defensive "never happen" case where the system still needs a defined behaviour.
- A list of self-inconsistencies. Combinations where the expert discovered they've been handling the case two different ways. Captured, not smoothed over.
- Test cases for free. Every column is a ready-made scenario; every don't-care collapse is a parameterised test.

Photograph the table from directly in front, lighting good enough to read every cell. Transcribe into a spreadsheet (spreadsheets are a Decision Table's native form) and share with everyone who was in the room.

These outputs feed straight into:

- Event Storming. Event Storming surfaces the hotspots, the places where the rule is tangled. Decision Tables are one of the tools you reach for to untangle a specific hotspot. Run Event Storming first to find the knot; run a Decision Table to unknot one of the knots it found.
- Example Mapping. Example Mapping is the close cousin. Use Example Mapping when you're trying to understand a rule through stories and questions; use a Decision Table when the rule is already understood well enough that you're trying to enumerate the cases. Many teams start with Example Mapping and graduate to a Decision Table when the rules crystallise.
- Threat Modelling. A Decision Table is a useful tool inside a threat model when the question is *"under what combinations of conditions does this become dangerous?"* The grid makes the dangerous cells literally visible.

Who's Needed

Three to four people plus the facilitator, sixty to ninety minutes:

- Facilitator. Does not contribute rules. Their job is to draw the grid, keep asking "and what about this combination," and prevent the expert from redesigning the rule on the fly instead of describing it.
- The domain expert (mandatory). Not "someone who knows a bit about it." The person who currently makes this decision, whether in their head, in a spreadsheet, or by calling a friend. If there are two experts who disagree, that's gold; bring them both and let the table surface the disagreement. For an SRE runbook, this is the on-call engineer who has actually made the call at 3am.
- One or two developers: the people who will implement the rule or who have been getting it wrong. Their job is to ask "what about…" until they run out of combinations. Developers who've been on the receiving end of the bugs are the best participants here; they arrive with scar tissue, and scar tissue asks sharper questions.
- A tester, if you have one. Testers think in edge cases as a first language. They will find the cells the developers didn't.

This is knowledge extraction, not brainstorming. More than five people and the session turns into a committee trying to redesign the rule.

Who to leave out:

- Stakeholders who want to change the rule. Decision Tables capture rules as they are. If someone wants the rule changed, that's a separate conversation and should happen *after* the table exists. A stakeholder who keeps saying "well, it should be…" will derail the session.
- Managers who feel responsible for the expert's answer. The expert needs to be able to say "I've been doing this wrong for two years" without worrying about who's listening.
- Spectators. This is a small, focused session. Everyone in the room should be asking questions.

How To Run It

Phase	Duration	Materials	Key question
Frame the decision, draw the grid	10 min	Whiteboard	"What are we deciding? What's the question?"
Identify the conditions	15 min	Whiteboard	"What factors does the answer depend on?"
Identify the outcomes	10 min	Whiteboard	"What are the possible answers?"
Fill the columns	30 min	Whiteboard	"What's the answer for this specific combination?"
Find the gaps	15 min	Whiteboard	"What combinations haven't we covered?"
Wrap-up, owners, next steps	10 min	,	"Which gaps need a policy decision?"
Total	90 min		

The active work is about 80 minutes; the remaining ten is the wrap-up. Keep the grid erasable; you will rewrite it at least once as new conditions surface.

The session is a conversation between the expert and the grid. The facilitator mediates: they ask the expert for the answer to a specific combination, write it in the cell, and then ask the developers whether that answer surprises them. Developers drive the gap-finding; the expert rarely discovers their own gaps, because the gaps are things they've never had to think about. The tester, if present, is the one who asks the question that makes the expert pause.

The rhythm is question, answer, write, next question. Keep it moving. The worst failure mode is a 15-minute debate about whether two conditions are really the same condition; park it with a question mark and fill in more cells.

Phase 1: Frame the decision, draw the grid (10 min)

Write the decision at the top of the whiteboard as a question. Not a statement, a question.

> *"What box does this subscriber receive?"*
>
> *"Can this deployment proceed?"*
>
> *"Should this alert page the on-call engineer?"*

Then draw the empty grid: two horizontal regions separated by a thick line. The top region is for conditions (inputs), the bottom for outcomes (outputs). Leave lots of columns: eight to start, more if you need them.

What to say:

> *"At the top we're going to list the factors that affect the answer. At the bottom we're going to list the possible answers. Each column is one specific combination, and we're going to fill in every column the rule actually produces. The goal is: by the end, no combination can come up and surprise us."*

What to watch for:

- The expert wanting to explain the rule first. Gently decline. *"Let's build the table and let the rule explain itself. I'll stop you and ask for specifics."*
- The decision being too big. If the question is "what should our pricing strategy be," you're at the wrong level. Narrow it: *"What price do we quote for this specific subscription type?"*
- The decision being too small. If there's clearly only one condition, you don't need a table. Say so and end the session early.

Phase 2: Identify the conditions (15 min)

Ask the expert:

> *"What does the answer depend on? Give me the factors, one at a time."*

Write each factor as a row in the top region. For each factor, establish the values it can take:

- Binary (yes/no): "Is the subscriber new?"
- Small discrete set: "Box size: Small, Medium, Large"
- Bucketed continuous value: "Delivery distance: under 50km / 50-100km / over 100km"

Continuous values must be bucketed. "How far away is the customer" is not a condition until the expert says where the thresholds are.

What to say:

> *"Can you give me a short name for that condition? And what values can it take? Two? Three? We'll write them down."*

> *"That condition sounds continuous. Where are the breakpoints? When does a 'short distance' become a 'medium distance'? The expert sets the thresholds, not the developer."*

What to watch for:

- Too many conditions. Six fully-independent binary conditions is 64 columns (unworkable) but in practice most pairs aren't fully independent. Once don't-care cells absorb the unused dimensions, a six-condition rule typically collapses to 12-20 columns. Don't enumerate 2^n upfront; capture rules as they come and let don't-cares (`-`) collapse the noise. If your table still doesn't collapse, split it or nest it.
- A condition that's really an outcome. *"We send them a welcome box"* is not a condition. If the expert names it as a factor, redirect: *"That sounds like the answer to a different combination. Let's put it in the outcomes row."*
- Missing conditions. The expert names three; a developer asks *"what about paused subscribers who've come back?"* and the expert pauses. That pause is a new condition. Add the row.
- Conditions that depend on context the table doesn't capture. If the answer depends on the weather, *add weather as a condition*. Don't decide the rule is "too fuzzy"; the whole point is to surface the fuzz.

Phase 3: Identify the outcomes (10 min)

Ask:

> *"What are the possible answers? Not the combinations, but the set of answers the rule can produce."*

Write each outcome as a row in the bottom region. Outcomes are typically one of:

- A choice from a small set: "Standard box / Welcome box / Curated box / Skip this week"
- A yes/no action: "Proceed / Block"
- A routing: "Page on-call / Send email / Log only / Ignore"
- A computed value with a small number of branches

What to say:

> *"Is there an 'error' or 'escalate to a human' outcome? Because if a combination comes up that we didn't expect, something has to happen. Don't leave that row empty."*

What to watch for:

- "That can't happen" outcomes. Add a "flag for review" or "error" outcome anyway. Software will encounter impossible combinations at 2am and the outcome needs to be defined, not discovered.
- Binary outcomes pretending to be multiple. If the rule is "proceed" or "don't proceed," you only need one outcome row with yes/no values.

- Outcomes that are too specific to one combination. *"Send a custom welcome box to new subscribers during peak season"* is three conditions compressed into one outcome. Pull it apart.

Phase 4: Fill the columns (30 min)

This is the core of the session. Start with the most common case:

> *"Regular subscriber, preferences set, off-peak, medium box. What do they get?"*

Write the values down the column, write the outcome at the bottom. Then vary one condition at a time:

> *"Same case, but they're a new subscriber. What changes?"*
>
> *"Same case, but it's peak season. What changes?"*
>
> *"Now what if they're new and it's peak season?"*

Work systematically. The expert's brain answers best when it can anchor on a previous case and change one thing. Don't jump around; walk the combinations.

What to say:

> *"Let's start with the most normal case and then break it one piece at a time."*

> *"What if someone came to you with this exact combination right now: what would you tell them?"*

> *"You paused. What were you thinking about? Tell me what's going through your head."*

What to watch for:

- "It depends" inside a column. If the expert says *"well, it depends on..."* while you're filling in a cell, you've found a missing condition. Add a new row to the top, rewrite the affected columns.
- The expert designing instead of describing. *"Well, it should probably..."* is a redesign. Redirect: *"What does the rule do today, not what should it do? We can talk about changes after the table is complete."*
- Contradictions between columns. *"New subscribers get a welcome box"* and *"peak season overrides everything"*: what happens to new subscribers in peak season? The table forces the answer. The expert may discover they've been doing it inconsistently. That's the whole point. Write down whatever they actually do today, and flag it.
- Don't-care cells. If an outcome is the same regardless of one condition's value, mark that cell with a dash. Don't-cares collapse the table and make it readable.
- The self-inconsistency moment. The expert says *"oh, I've been doing that wrong for a year."* Capture it. Do not smooth it over. That discovery was the ROI of the session.

Phase 5: Check completeness and consistency (15 min)

Two distinct properties to test, in this order.

Completeness: every input combination has an answer. Step back. Count the columns. With four binary conditions the maximum is sixteen; if you have twelve columns, which four are you missing? Go find them. Walk the edge cases people brought into the room. For each, find its column. If the table answers it correctly, the rule is captured. If not, adjust.

> *"Is there any combination we haven't written down that could actually occur? Any combination where nobody in this room knows what we do today?"*

Those are the gaps. Mark them clearly. Each one is either (a) a policy decision somebody needs to make, or (b) an "impossible" combination that still needs a defensive outcome.

Consistency: no two columns whose conditions overlap can give different outcomes. Once you allow don't-cares, two rules can match the same input. *"New subscriber with promo code"* might match both the "new subscriber" rule and the "promo active" rule, and they might have different answers. Walk every pair of columns whose condition cells overlap and check the outcomes match. Where they don't, declare a hit policy (*Unique*, *First*, *Priority*, or *Any*; see *Definitions & Background*). Pick one and write it on the wall before Phase 6, or you'll discover the conflict in production.

What to say:

> *"If this combination walked through the door tomorrow, what would we actually do? Not what should we do; what would happen?"*

> *"You said that combination can't happen. What would you do if it did? Because software will find it."*

What to watch for:

- The expert changing the rule mid-review. *"Actually, looking at this, new subscribers in peak season should get…"* That's a policy decision, and it's a real one, but it's not today's rule. Capture both: the current behaviour and the proposed new behaviour, clearly distinguished.
- Dismissed impossibilities. *"That combination can't occur."* Push. *"What would the system do if it did? Crash? Silently do the wrong thing? Page someone?"* The defensive answer goes in the cell.
- Gaps that need authority. Some policy decisions belong to someone who isn't in the room. Flag those; don't invent answers.

Worked example

See Decision Tables: Making Maya's Brain Explicit, where the Greenbox team walks a domain expert through ninety minutes at a whiteboard, building the grid that had been living in her head for two years. The moment the rule she'd been handling inconsistently becomes visible as a specific column is the moment the pattern earns its cost.

What Can Go Wrong

The rule designer. The expert starts redesigning the rule instead of describing it. *Recovery: "Let's capture how it works today first. We'll talk about changes once we can see the whole picture." Stop if:* After three redirects the expert still can't describe the current rule, only their ideal version. You're in a policy session disguised as a capture session. End early and rebook as a policy workshop.

The explosion. Too many conditions, the grid is past twenty columns, nobody can read it. *Recovery:* Look for conditions that always move together (they're really one condition) or split into two sequential tables (decision A first, then decision B using A's output as input). *Stop if:* You can't simplify and the table is at forty columns. The decision is really a family of decisions; reframe as a Policy Level session and plan it for half a day.

The “I just know” expert. The expert can make the correct call every time but can’t articulate the rule in the abstract. *Recovery:* Work from concrete cases instead. *“Last Wednesday, this specific subscriber came up. What did you do? Why?”* Walk the table by example rather than by combination. *Stop if:* The expert can’t reconstruct any specific recent case. You need a different expert, or you need to shadow this one for a week and come back.

The perfectionist. Someone wants every cell exactly correct before moving on, and forty minutes have passed on one column. *Recovery: “Put your best answer in and mark the cell with a question mark. We’ll come back.”* Speed through the table first; polish later. *Stop if:* The perfectionism comes from a real blocker: the expert genuinely doesn’t know, and nobody in the room does either. That cell is a policy gap. Mark it and move on.

The derailer. Someone keeps raising rules that belong in a different table. *Recovery: “Great catch. Write it on a sticky and we’ll schedule a separate session.”* Keep one table per session. *Stop if:* Every other comment is about a different decision. The scope was wrong; the rules are tangled and need a Policy Level session to separate them.

The silent expert. The expert has gone quiet and the developers are guessing. *Recovery:* Check in. *“You’ve gone quiet; what are you thinking?”* Often the expert has realised the rule is inconsistent and is embarrassed to say so. Name it: *“If the answer today is ‘we’ve been doing it two different ways,’ that’s a finding, not a failure.”* *Stop if:* The silence is political: the expert doesn’t want to document the rule because documenting it will expose something. End the session and deal with the politics separately.

Next Steps

The session ends; the work begins.

Same day, the facilitator:

- A high-resolution photograph of the completed table, lighting good enough to read every cell.
- Transcribe the table into a spreadsheet. Spreadsheets are a Decision Table’s native form. Share it with everyone who was in the room.
- A short summary to participants: *“Here’s the table. Here are the gaps we flagged. Here’s who’s deciding what.”*

This week, the product owner:

- Triage the gaps. Each flagged cell is one of: (a) a policy decision the product owner can make, (b) a policy decision that needs authority from outside the team, (c) a defensive “never happen” case where you need to define what the system does anyway. Resolve (a) this week; escalate (b) to whoever owns the policy; document (c) as a guard in the backlog.
- Turn the table into tests. Every column is a test case. Every don’t-care collapse is a parameterised test. Hand the spreadsheet to the developer implementing the rule; they will thank you.
- Walk the table past a second expert if one exists. Different experts often apply different rules. If a second expert reads the table and says *“that’s not what I do,”* you have a second finding: the rule isn’t one rule.

- Write the policy one-pager. If the session surfaced material policy questions, draft them for the person who decides, with the table as evidence.

Ongoing, the team:

- When the rule changes, update the table first, then update the code. The table is the specification.
- When a new edge case appears in production, check the table. If the table covers it and the code doesn't, it's an implementation bug. If the table doesn't cover it, the table needs a new column.
- Keep the table in the same place as the code it governs: in the repo, alongside the tests, so it goes stale only when the code goes stale.

Variants

Standard (default). One business decision with 4–6 interacting conditions, sixty to ninety minutes, three to four people plus facilitator. Output: a complete grid, policy gaps surfaced, test cases ready. This is what most rules need, and the rest of this post describes it.

Rule Level *(zoom in)*. A single decision with 2–3 conditions, 30–45 minutes. Output: a small table, sometimes just a cleanup of an existing `if`. Reach for this when you suspect a small rule is hiding a bigger problem; often the table reveals that the "small" rule has more conditions than anyone admitted.

Policy Level *(zoom out)*. A family of related decisions across a process, half a day or split across two sessions. Output: multiple linked tables, often a policy review afterwards. This is what you reach for when the whole team is arguing about pricing, or when an SRE runbook has grown into a jungle. Expect to schedule a follow-up policy review with whoever owns the rules; Policy Level surfaces decisions, it rarely closes them.

Remote. A shared spreadsheet with the conditions as rows, outcomes as rows below a separator, and combinations as columns. Slower than a whiteboard (the rhythm of *"answer, write, next"* is faster in person) but the structure transfers cleanly. Use one shared cursor: only the facilitator types, prompted by the room, to keep the grid legible.

Two-expert variant. When two domain experts apply different rules to the same decision, run the session with both in the room. Don't try to resolve the disagreement live; capture each expert's answer in parallel rows or a parallel column, and let the table surface the divergence. The output is two tables, not one, and the next conversation is a policy decision about which one is the rule.

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-decision-tables/.

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