

bardd User Guide

Version 1.0.8 · 2 November 2026

A Codemanship application · www.codemanship.co.uk · © Codemanship Ltd 2026

In Responsibility-Driven Design you start with what has to happen, not with who does it. This guide covers bardd 1.0.8 (Behaviour And Responsibility-Driven Design), a responsibilities-first workbench: you put each outcome in your Gherkin specs on a card as a responsibility, name the cards once you can see what they do, then realise the scenarios as sequence diagrams to find out who works with whom. As the design grows, the Review tab points out smells, and model refactorings reshape the design without changing what it does.

The process at a glance

You find the design by asking what has to happen before asking who does it. Names come once each card shows what it is for, and collaborations come from acting out scenarios, not from guessing.

Responsibilities first, names next, then scenarios



The RDD process in the workbench · 6 stages

1. **Start from the specs.** Each **When** step is a request into the system. Each **Then** step is an outcome the system must deliver. Both become responsibilities: something that will be done.
2. **Assign responsibilities.** For each step, say what has to happen ("Calculate order total") and what it needs to know to do it (products, prices, discounts). At this point you don't know who does it, and you don't need to. *Something* will.
3. **Group them on cards.** Put the responsibility on a new, unnamed card, or on a card that already knows what it needs. This is Information Expert: the job goes to whoever has the information. Each card gathers a few things it does and the things it knows to do them, as on a blank index card in a face-to-face workshop.
4. **Name the roles.** Once a card is full enough to show its purpose, find the best name for it, preferably in the language of the spec. Naming is the checkpoint: you can't act out a scenario with a card nobody has named, so only named cards go into sequence and class diagrams.

5. **Realise the scenarios.** Act out each scenario as calls between roles. Each call asks another role for one of its responsibilities, and that makes the caller and receiver collaborators. When a call needs something no card does yet, go back and add it.
6. **Review and refactor.** Look for smells (too many responsibilities or dependencies, cycles, envy) and check how closely the design's words match the spec's. Reshape the design with refactorings: they move responsibilities and roles around while every outcome you've realised stays realised. Then realise the scenarios again.

The workbench's tabs follow the same order: **Specs** (stages 1 to 4), **CRC cards**, **Sequence**, **Class diagram** and **Review**. Each tab has an icon; when the window is too narrow for the names, only the icons show.

Watch the demo

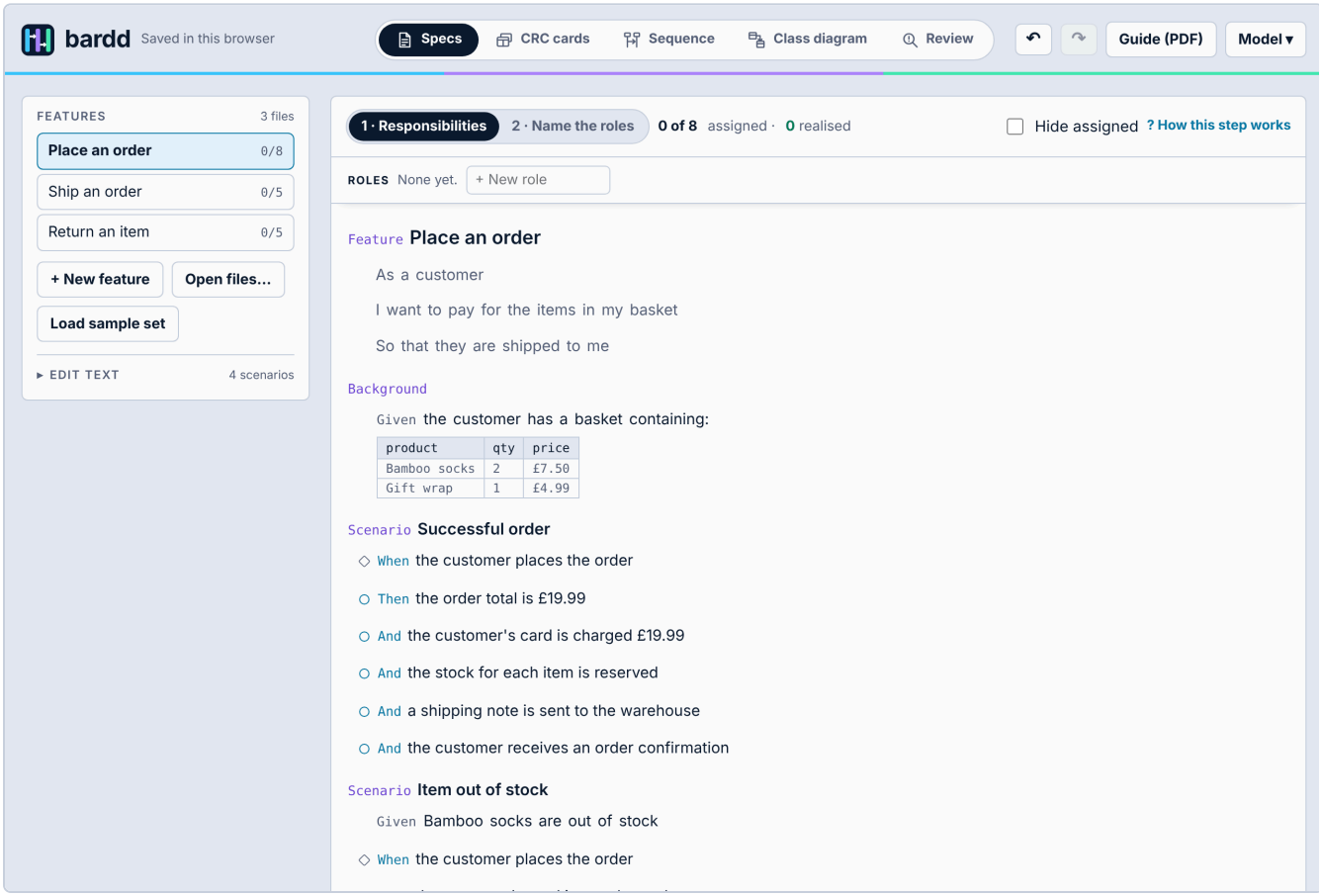
Click **Demo** at the top of the screen to watch bardd design a small online shop from the sample features, from the first responsibility to a refactored design. It takes 71 steps in five chapters, and clicks the buttons for you while a panel explains each step.

Chapter	Steps	What it shows
1 Assign responsibilities	1–17	Each outcome of Successful order gets a responsibility on a role with no name yet, with what it needs to know
2 Name the roles	18–30	Order, Basket, Card, Item, Warehouse and Customer, from the suggestions and the spec; then the CRC cards, and Price added to Item from the Background's table
3 Realise the scenario	31–50	The Successful order sequence diagram: calls, returns, getters, multiplicities and two loops, until all five outcomes are realised
4 The class diagram	51–53	How Knows, Does and collaborators become attributes, operations and associations
5 Review and refactor	54–71	Four smells; Extract "checkout" to Basket, Extract "subtotal" to Item, Inline the getters; no smells left

- **Next** (→ or Enter) does the step: a pointer moves to the control the panel describes, which is ringed, and clicks or types. **Back** (←) puts the step before back exactly as it was. **Exit demo** (Esc) stops at any point; the last step says **Finish**.
- The panel sits at the bottom right, or the top right when that would hide what it explains. Drag it by its blue header to move it; it stays there until the demo ends.
- On the Sequence tab, the diagram scrolls to the call each step adds or changes.
- The demo runs on a model of its own, and the header says **Demo: nothing is saved**. When it ends, your own model, tab and undo history come back exactly as they were.
- While it runs, clicks and keys don't reach the page, and messages at the bottom of the screen go after 2 seconds.

1. Load your feature files

Start on the **Specs** tab. The left-hand panel lists your feature files; the right-hand panel shows the one you're working on. After a short loading screen, shown once per browser session, the workbench opens empty, ready for your own features; the sample set is there if you want to practise first.



- **Load sample set** loads three related features to practise on: Place an order, Ship an order and Return an item.
- **Open files…** loads `.feature` files from your computer. You can pick several at once, or drag them onto the panel. A file with the same name as one already loaded replaces it.
- **+ New feature** gives you an empty feature to type or paste Gherkin into.
- Click a feature in the list to work on it. The numbers beside it (for example 5/8) show how many of its outcomes you've assigned.
- **Remove** takes the feature you're editing out of the model. Anything already on the cards stays.

Edit text, under the list, opens the Gherkin of the feature you're working on, to read, type or paste. Its heading shows how many scenarios the feature has, and any problems found reading it. It's folded away while you work, so the spec on the right gets the full width; it opens by itself for a new, empty feature.

Each feature needs its own `Feature:` name. If two files share one, you'll see a warning under the text box. Related features share one set of cards, so a card built for one feature is there to reuse in the next.

2. Assign responsibilities

The Specs tab works in two steps, chosen with the switch at the top of the spec: **1 · Responsibilities** and **2 · Name the roles**. It opens in step 1. Here you go through the spec line by line, saying what has to happen and putting it on a card. Nobody has a name yet.

The bar above the spec shows how far you've got across all features, for example 6 of 8 assigned · 4 realised. Tick **Hide assigned** to see only what's left to do. **How this step works** repeats these instructions on screen.

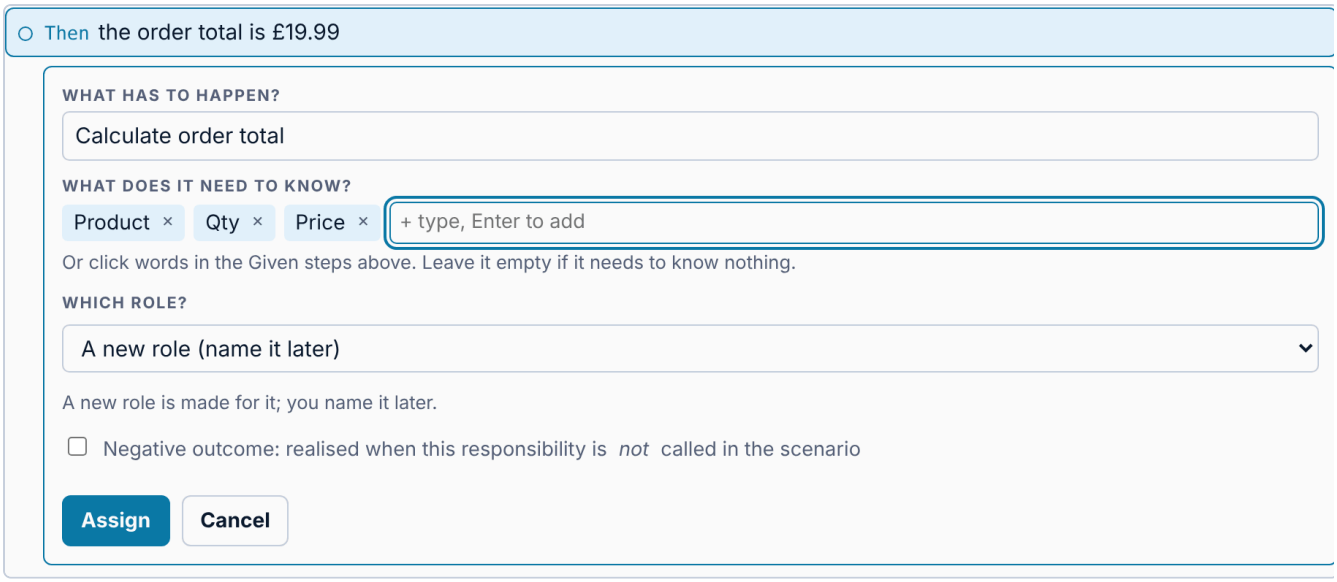
What you click decides what you add:

You click	It becomes	Counts towards progress
A **When** step	Something a card does, asked for by the Client	No
A **Then** step, or an **And** or **But** after it (an outcome)	Something a card does	Yes
Words in a **Given** step, or a table heading, while no form is open	Something a card knows	No

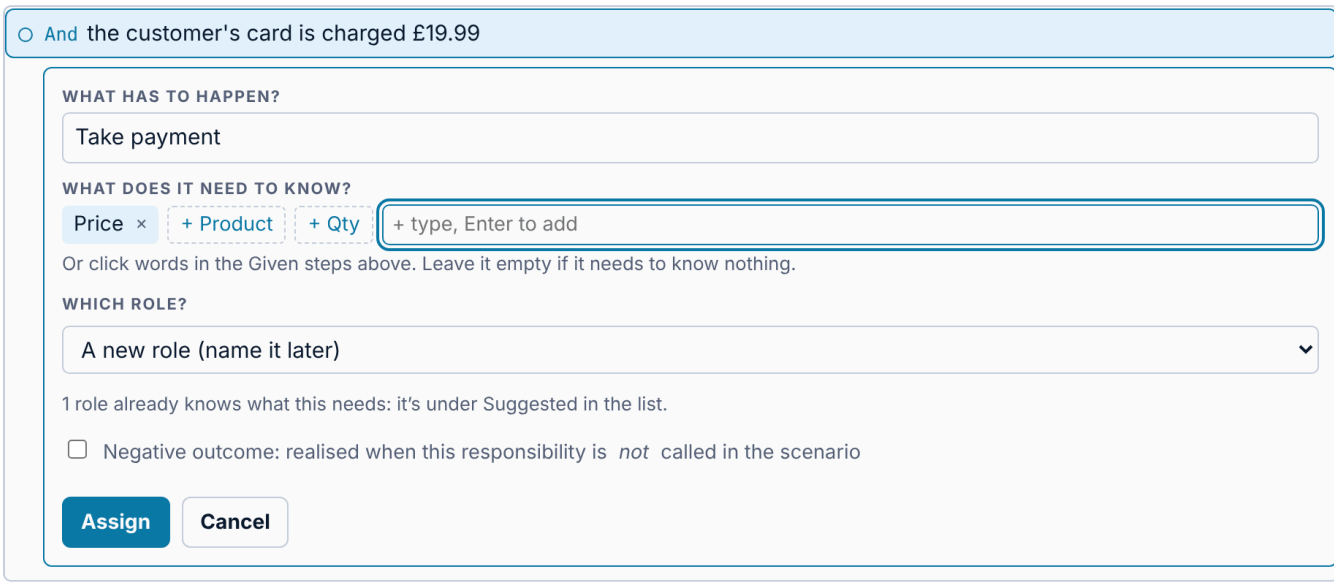
Assigning a When step or an outcome. Click it, and a form opens beneath it with three questions, in order:

1. **What has to happen?** Name the responsibility as something done, such as "Calculate order total". If it could be one you already have, it's suggested underneath; click a suggestion to use it.
2. **What does it need to know?** Add the information this responsibility uses. Click a suggested chip (taken from table headings in the Given steps, including the Background, and from things cards already know that the scenario mentions), click or drag across words in a Given step, or type and press Enter. Leave it empty if it needs to know nothing.
3. **Which role?** Pick it from the list. **A new role** is chosen unless you pick another; you name it later. Roles that already know what you said it needs are grouped under **Suggested** at the top of the list, marked for example *knows Price*; the rest follow under **Other roles**. The line under the list describes the role you chose by what it does and knows. Giving the job to the role that has the information is Information Expert.

Then click **Assign**. The form closes and leaves you on the outcome you just assigned; click the next outcome when you're ready. Cancel or Esc closes the form without assigning anything.



What it needs is added to the card as things it Knows, and remembered against that responsibility, so the card shows *needs Product, Qty, Price* under it.



When steps are entry points. A When step is a request into the system, so the card you give it is the one the Client calls first in that scenario's sequence diagram. The same When step often opens several scenarios; the form offers to assign all the identically worded ones in the feature at once, ticked by default. When steps show a diamond marker and an **entry** tag, and don't count towards progress.

Reuse a responsibility whenever it fits. One responsibility often delivers several outcomes, across scenarios and features. Pick it from the suggestions, or choose the role that already does it (it's marked **already does this**).

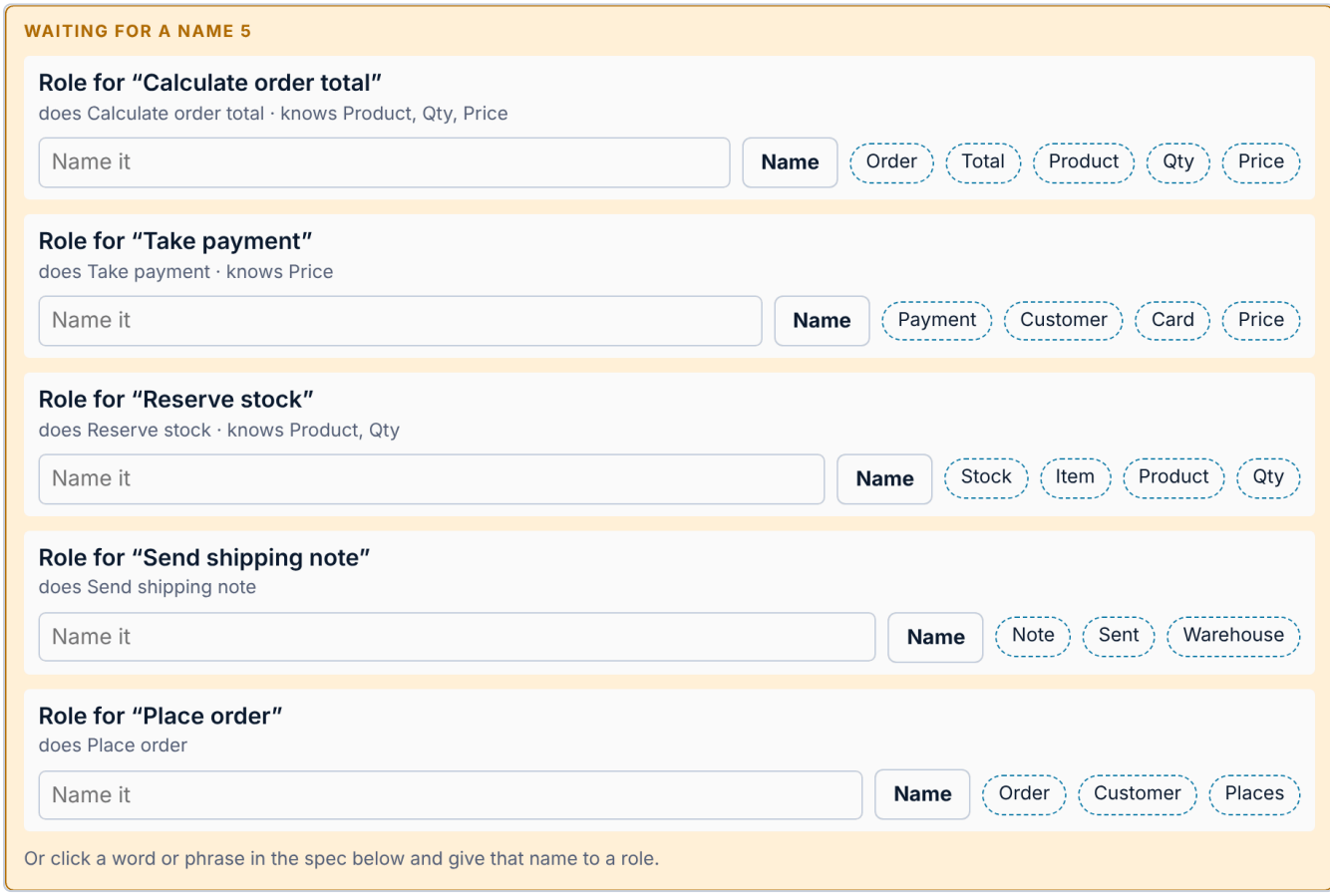
Negative outcome is optional. Tick it when a Then says something *doesn't* happen, such as "the card is not charged". The chip then shows **NOT** before the responsibility, and the outcome counts as realised when the scenario's sequence diagram does *not* call it. Click ¬ on a chip to switch it on or off later.

Assigned lines show a chip with the role and responsibility. A role with no name is shown by its first responsibility, such as *Role for “Calculate order total”*. A ✓ on the chip means the outcome is realised in its scenario's sequence diagram. Point at a chip to see its other controls: ¬ switches a negative outcome on or off, × unassigns it, and +2 means the same responsibility is linked to two other lines (hover to see which). Click an assigned line again to give it to a second role as well.

To add something a card knows without an outcome, close any open form and click words in a Given step or a table heading, then choose the role.

3. Name the roles

Once a card holds enough to show what it's for, give it a name. Switch to **2 · Name the roles**; the number on the switch is how many cards are still waiting for one. You can name cards at any time, and go back to step 1 whenever you like.



Waiting for a name lists every card without one, described by what it does and knows. For each card:

- Type a name and press Enter, or click **Name**.
- Or click one of the suggestions after the button. They're the words that come up most in the card's outcomes, its responsibilities and what it knows, so they come from the spec.

Names stay as you'd say them: "shipping note" becomes "Shipping note". Code-style names such as `ShippingNote` appear only in the class diagram. Two cards can't have the same name; if they turn out to be the same role, fold one into the other with Inline role (section 8).

Naming from the spec. Click any word in the spec below the panel: the feature name, its description (“As a customer…”), a scenario name, any step, or a table heading. For a phrase, drag across the words, or click the first word and then the last. A form opens with the name. Under **Give this name to**, click the role it names. Or click **Make it a new role** for a role that doesn't exist yet, or **Link to …** when a role already has that name. Clicking a role in the **Roles** tray does the same, and **+ New role** in the tray adds a role by name. A small word on its own, such as “a” or “has”, can't name a role.

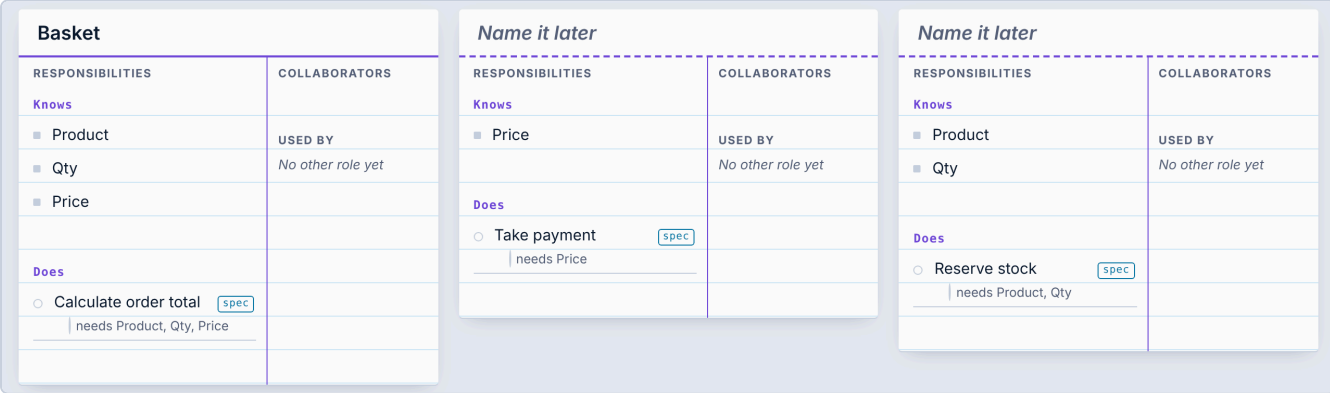
Each role in the **Roles** tray has a small card icon, a reminder that every role is a CRC card on the CRC cards tab.

In this step, a word that names a role is **shaded**, like a highlighter, and a phrase you've linked to a role is shaded more darkly, so you can see who the spec talks about (the key is in the Roles tray). When nothing is selected, click a role in the tray to highlight where it appears in the feature: every mention, and its assignments, get a blue background with an underline. **Next** steps through them; click the role again, press Escape or click **Clear** to stop. The tray highlight works in step 1 too.

A good name is one the people who wrote the spec would recognise. The Review tab (section 7) shows how many of your words come from the spec.

4. Refine the CRC cards

The **CRC cards** tab shows every card: what it **Knows**, what it **Does**, and its **Collaborators**, the roles it asks for help. The line at the top counts the cards still to be named.



- **Cards without a name** have a dashed line under the title and say *Name it later*. Click the title and type to name one here instead of in the Specs tab.
- **Needs:** under a responsibility, *needs Product, Qty, Price* lists what you said it needs to know. Renaming one of those Knows items, or the responsibility, keeps the list up to date.
- **Edit anything in place.** Click a name or item and type. Press Enter when you've finished typing; use + Knows or + Does to add another item. Renaming an item keeps its links to the specs, and renaming a Does item also renames every call to it in the sequence diagrams.
- **+ Knows**, **+ Does** and **+ Collaborator** add to a card. They appear on the card you're pointing at or working in, so the rest stay quiet; so do × to remove an item and **+ Type**. **+ New card** adds a card. Long items wrap, so nothing is cut off.

- **Click a name** under Collaborators or Used by to jump to that card.
- **Refactor:** ⇥ beside a Does item moves it to another role, and ⋯ in a card's header offers **Extract role…** and **Inline role…**. These keep the scenarios working while they reshape the cards; see section 8.

Collaborators mostly come from the sequence diagrams (section 5), but you can add them by hand. A dotted name isn't used by any call yet. Removing one that calls still use asks you to confirm first.

Used by, under the collaborators, lists the roles that have this one as a collaborator: the other side of the same relationship. You can't edit it there; it changes when those roles' collaborators change. Click a name to jump to that card.

Multiplicity: the small field after each collaborator says how many of them the role works with. A grey **1** is the default. Type another value, such as **0..1**, * or **1..***, to change it, or clear it to go back to 1. It's the same value as the multiplicity at that end of the class diagram, so changing either changes both.

Types: when several roles are played by the same kind of object, such as a customer and a manager who are both people, hover a card and click + **Type**. Enter the type and press Enter. The card then shows "played by **Person**". Roles of the same type share one class in the class diagram (section 6). Everything else, including collaborators and messages, stays with the roles.

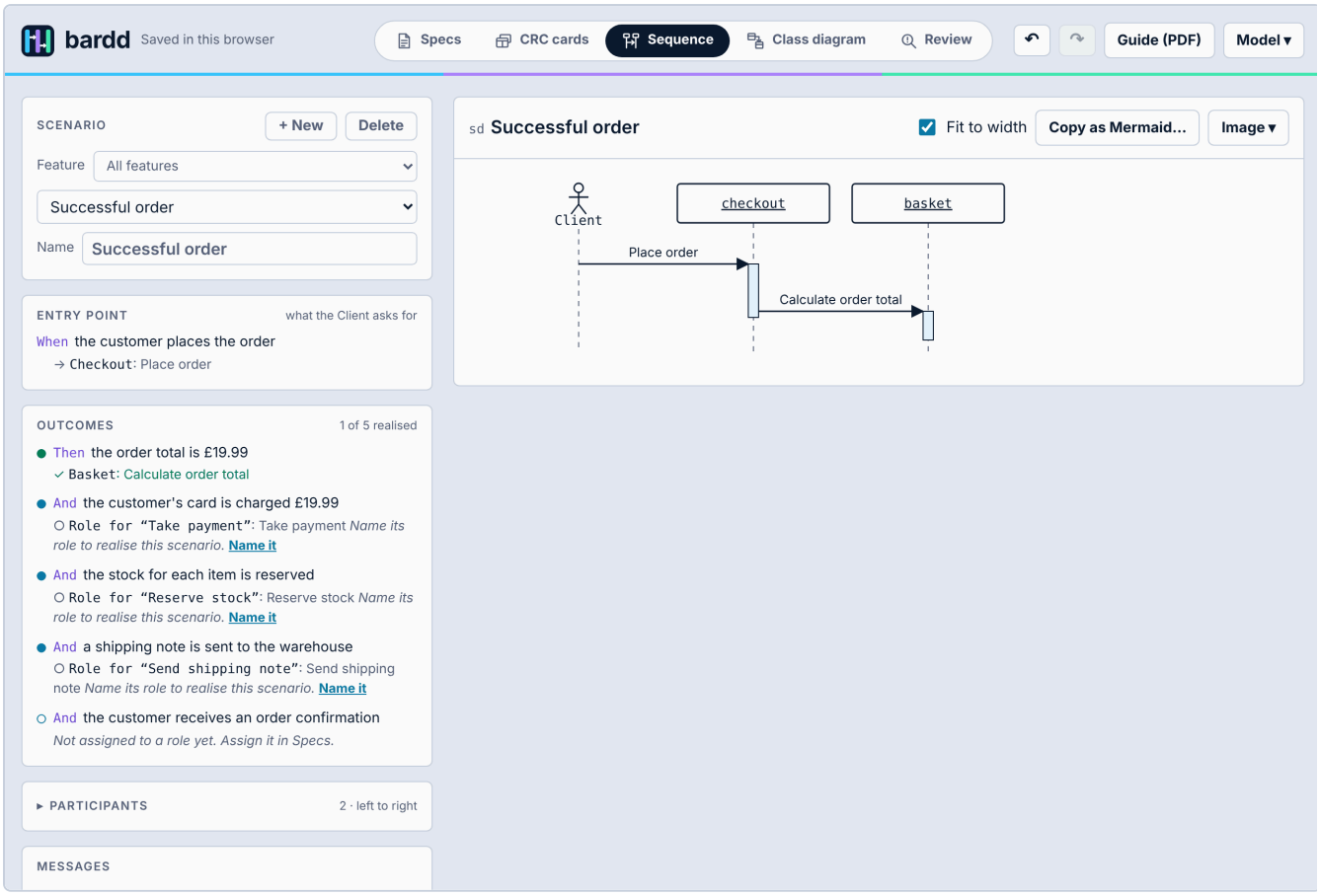
Merging: when two items turn out to mean the same thing, hover over one and click ⤵, then choose the item to merge it into. Its spec links and sequence messages move with it.

Deleting a whole card also deletes its messages in every sequence diagram, plus any message that depended on it, so no diagram is left with stranded calls. The delete button shows how many messages will go before you confirm. If you delete an item that was linked to an outcome, the Specs tab marks that link and offers **Re-add to card**.

5. Realise the scenarios to find the collaborators

The **Sequence** tab tests the cards by acting out a scenario as messages between roles. Each call asks the receiving role to do one of its responsibilities, and that is how collaborations are found: once a call asks for a responsibility, the receiver becomes a collaborator of the sender. That happens when you've finished with the message (you close it, open another one or leave the tab), so you can change its sender or receiver first without leaving a collaboration behind. The diagram on the right redraws as you edit.

Only named cards take part. You can't act out a scenario with a card nobody has named, so cards without a name aren't offered as participants, and the panel says how many are waiting. In the **Outcomes** panel, an outcome given to a card without a name says *Name its role to realise this scenario*; click **Name it** to open the card.



1. **Scenario:** choose a feature, then one of its scenarios. Scenarios marked "not drawn yet" come from that feature's Gherkin; choosing one starts its diagram, already holding the Client's calls for any When steps you've assigned to named cards. Or click **+ New** for a scenario of your own.
2. **Participants** (folded; its heading says how many): every diagram starts with **Client**, a stick figure standing for whoever asks the first role to act. Add the roles involved, left to right, with **+ Add a role…**. Type a name and click **Add new role** for a role that doesn't exist yet. A message's sender and receiver are chosen from these participants.
3. **Messages:** each message is one line in the list, such as *Checkout → Basket: List line items*, with returns in italics. Click a line, or its arrow in the diagram, to open it; both are highlighted in blue. In an open message, pick the sender and receiver, and choose **→ call** or **⇠ return**. A call asks the receiver for one of its responsibilities: choose it from the receiver's Does items, or pick **+ New responsibility for …** to add one to its card on the spot. Name a return after what comes back. **+ Message** adds one at the end, + inserts one straight after the open message, the arrows reorder, × deletes. **How messages work** repeats this on screen.

Getters. When the call is for something the receiver knows, look under **Add a getter** in the same list. It offers "get" and the name of each of the receiver's Knows items, such as *get price* for Item, unless the role already does that. Choosing one adds it to the receiver's card and uses it for the call, in one step. The Review tab counts these as getters, for Data class and Feature envy.

Each role's lifeline is labelled the UML way, as an underlined instance name starting with a lower-case letter: **checkout**, or **customer : Person** when the role has a type.

Who sends it. A new message is sent by whoever holds the baton at that point. A call passes the baton to its receiver, and a return hands it back. ⊣, **+ Message** and **+ Add call** all use this rule, so change the sender only when someone else should ask.

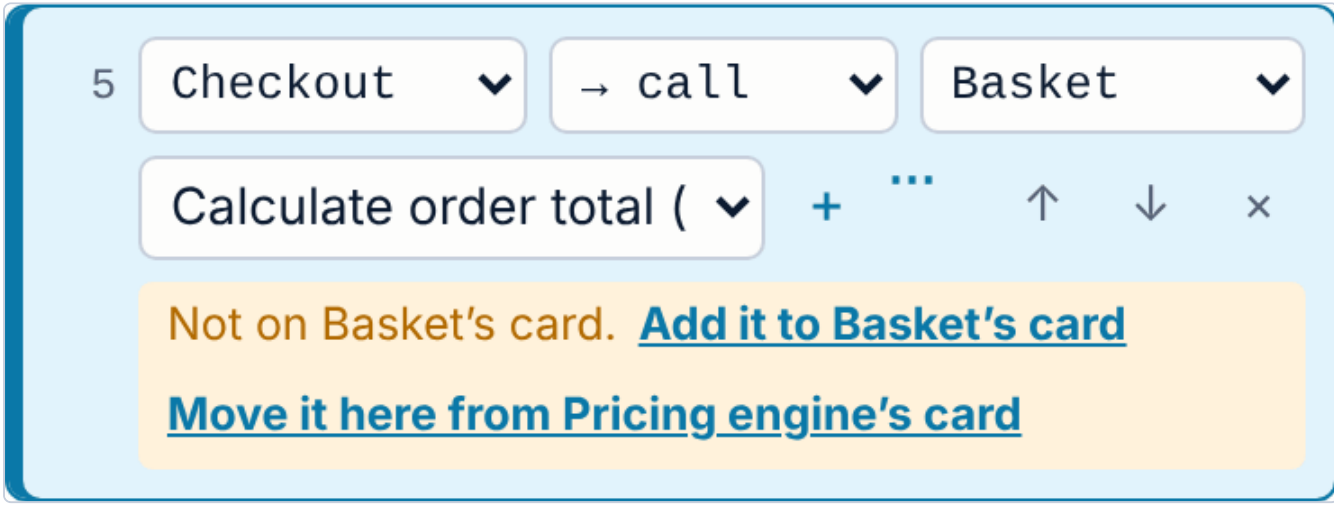
The Outcomes panel lists the scenario's Then steps and ticks each one off as the diagram realises it. Click **+ Add call** beside one that isn't realised yet to add the call it needs. If the diagram already ends with returns to the Client, the call goes in before them. **The Entry point panel** shows the scenario's When steps and their cards; click **+ Add Client call** if the Client doesn't make one of those calls yet.

Returns. Click ⇠ on a call to add its return: **Next message** puts it straight after the call, **Last message** at the end of the diagram. If the receiver makes calls of its own first, a third option puts the return after those. On the last message ⇠ adds the return straight away. The button only shows on calls that still need a return, so it also shows which are missing.

A role calling itself. A role can call itself for one of its own responsibilities. Draw that call's return (⇠ → *After the rest of …'s messages here*) and the call becomes a nested activation, as in UML: a second bar on the lifeline, holding everything the role does for it until the return. Without a return it's a short blip.

Repeating calls. When a role works with many of a collaborator (a multiplicity such as * or **1..***), its calls to that collaborator show ↻. Click it to repeat the call for each one. The diagram draws a UML loop frame around the call, its return and anything the receiver does in between; calls straight after it to the same receiver join the loop too. Above the first message in the loop, edit the condition ("for each line item"), click **Include next** or **Leave out last** to resize it, or × to remove it.

Changing who sends or receives a call. A call's return follows it: change the sender and the return now goes back to the new sender; change the receiver and the return comes from the new one. If the new receiver doesn't have the responsibility, the message says so and offers **Add it to …'s card** or, when nothing else still asks the old receiver for it, **Move it here from …'s card**, which takes it off the old card with what it needs to know. The change and the fix are one step, so a single Undo puts both back.



Unreachable messages. A message drawn in dotted orange has a sender that hasn't been called earlier in the scenario, so nothing asks it to act. Move a call to that role above it, or delete the message. Unreachable messages don't count towards realising outcomes.

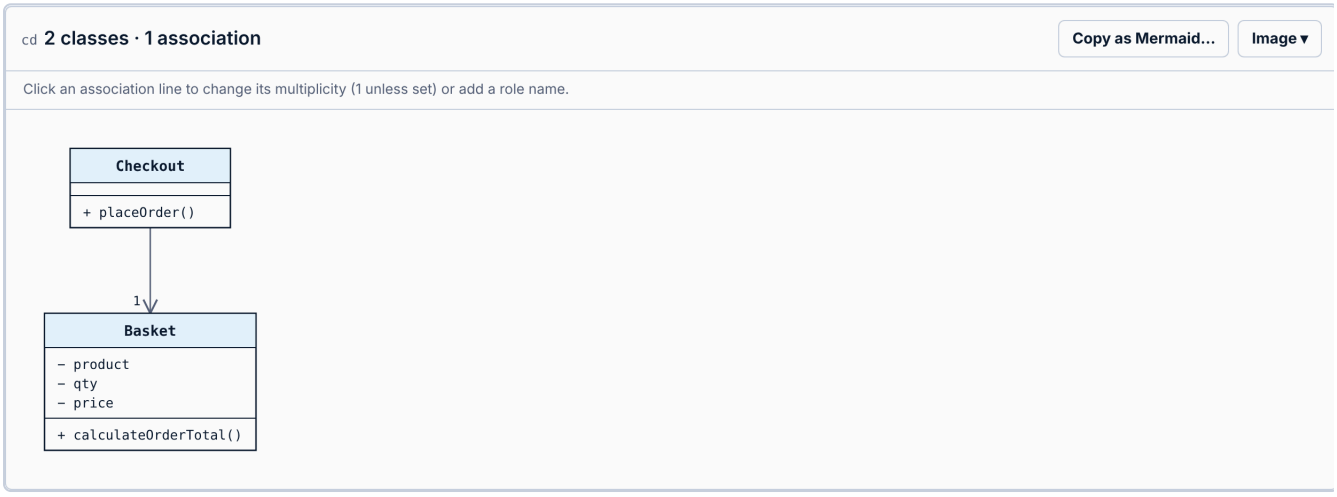
Collaborations and tidying up. Returns, and calls with no responsibility chosen yet, never add collaborators. A call you're still editing adds its collaborator when you close it, and the toast says so. A message can't ask for something on a Knows list: roles do things, they aren't queried for data. When a change leaves a collaboration unused in every scenario, such as correcting a sender or deleting a message, a prompt at the bottom of the screen asks whether to delete it. When a role is left with no messages in a diagram, it asks whether to remove it from that diagram. Answer **Delete it** / **Remove it** or **Keep** (or all at once when there are several). Undo and Redo keep the prompts in step: undoing the change clears its prompt, and redoing it asks again.

Fit to width shrinks the diagram to the panel. **Copy as Mermaid…** gives you the diagram as text for GitHub, Notion or slides. **Image ▾** copies it as a picture or saves it as an SVG or PNG file.

Refactoring from a message. ⋯ on an open call offers **Extract a responsibility…**, **Move “…”…** and **Inline “…”…**. They're described in section 8.

6. Review the class diagram

The **Class diagram** tab draws the structure your named cards add up to. Knows items become attributes, Does items become operations, and each collaborator becomes an arrow towards the role it calls. You change the design on the cards; this tab only shows it.



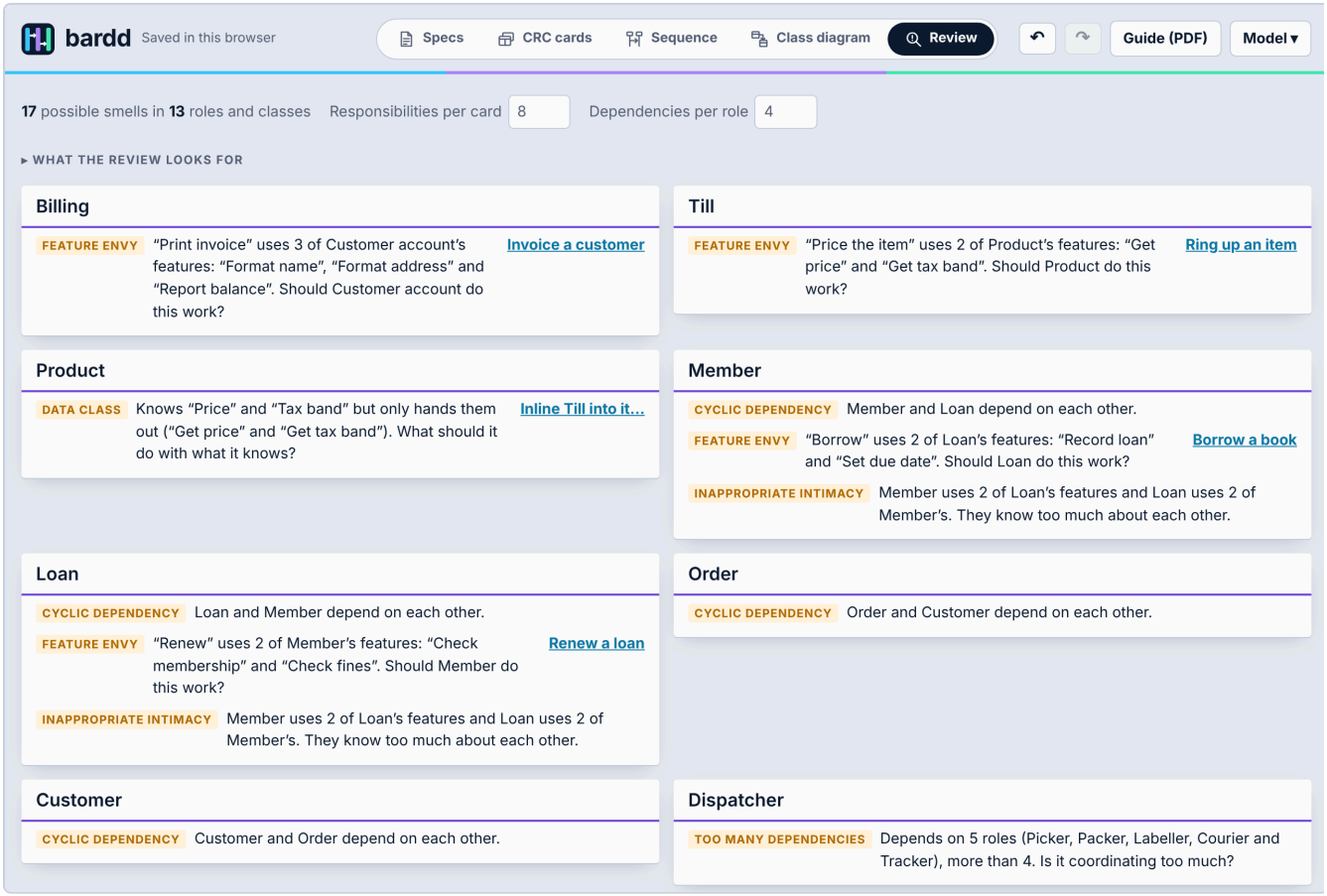
- **Cards without a name aren't drawn.** A class needs a name, so they're listed under the diagram instead (“2 roles aren’t shown until they’re named”), each linked to its card. The diagram fills in as you name them.
- **Grey operations** aren't called in any sequence diagram yet. That's worth a question: is the responsibility needed, or is a scenario missing?
- **Click an association line** to change the multiplicity at each end (for example 1, 0..1 or *) and add a role name at the end the arrow points to. It's the same value as on the CRC card's collaborator. Click **Done** to close it.

- **Click a class** to open its CRC card.
- **Code-style names** (on to start with) shows names as code (`takePayment()`); untick it for names as written (“Take payment”). **Fit to width** shrinks a large diagram to the screen.
- **Copy as Mermaid…** gives you the diagram as text, including multiplicities and role names. **Image ▾** copies it as a picture or saves it as an SVG or PNG file.

Types. Roles of the same type (section 4) share one class, with their attributes and operations combined. Each pair of collaborating roles still gets its own line, and a role's name appears at its end of a line when it differs from the type there. Clicking a shared class lists the roles that play it.

7. Review the design

The Review tab looks over the whole model for design smells, then compares the design's vocabulary with the spec's. Its findings are prompts for discussion, not verdicts: each one is a question worth asking about the design. Roles without a name are reviewed too, shown by their first responsibility; a role is cheapest to reshape before it's named. Review the design whenever a few scenarios are realised, not only at the end.



Smells. Each finding names the role (or the class, for roles that share a type), the smell, and a sentence explaining what was found, such as *“Print invoice” uses 3 of Customer account’s features... Should Customer account do this work?* **What the review looks for**, at the top of

the tab, explains each smell. Set your own limits in **Responsibilities per card** (8 to start with) and **Dependencies per role** (4); they're remembered in this browser. The last column below is the refactoring that usually helps (section 8).

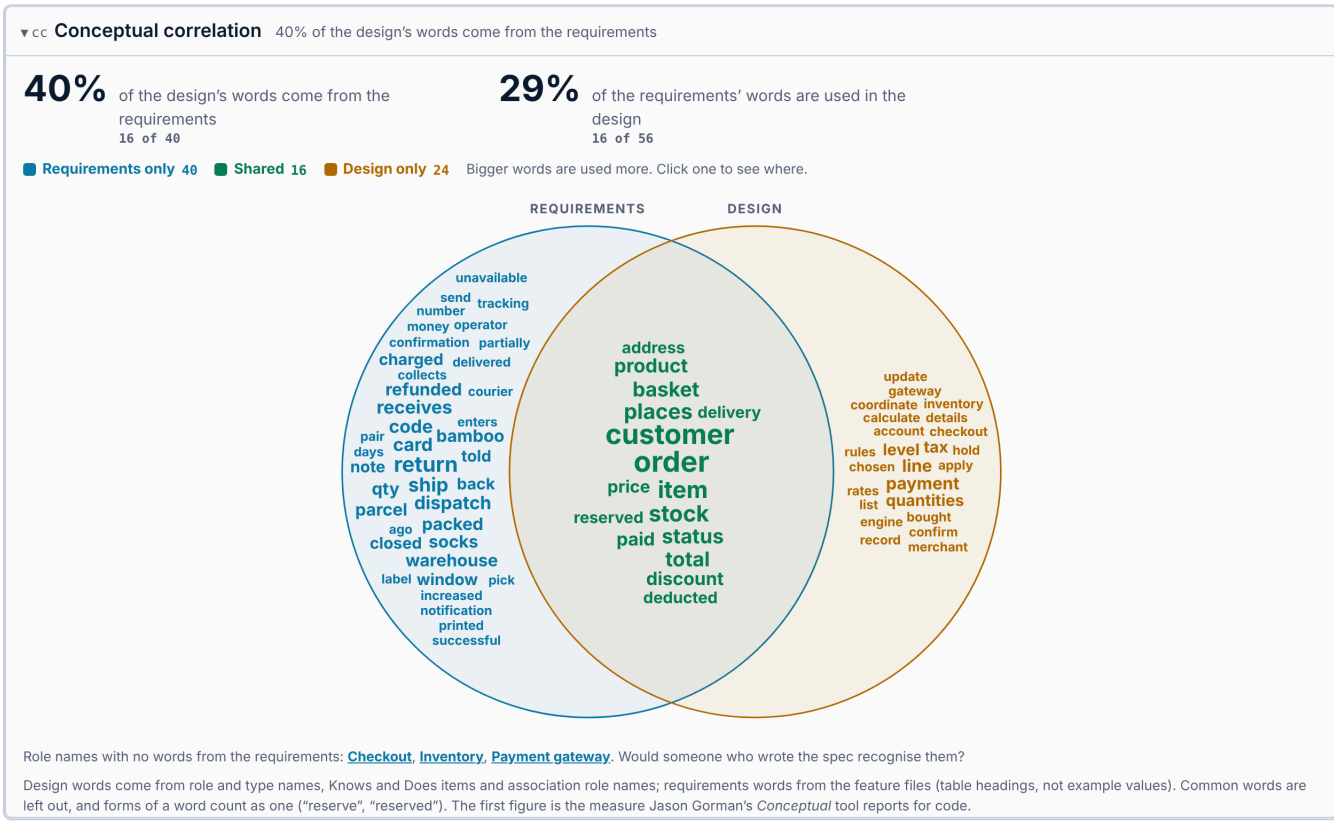
Smell	What the workbench looks for	What usually helps
Too many responsibilities	More Knows and Does items than fit on a card	**Extract role**: split off the responsibilities that work on the same information
Too many dependencies	Depends on more roles than the limit	**Extract responsibility** into a new role that coordinates some of those calls
Cyclic dependency	Roles that depend on each other, directly or round a loop	**Move** one side of the cycle to a role that depends on neither
Feature envy	A responsibility that uses two or more different features of the same other role	**Extract responsibility** into the role it envies; if all of it is envious, **Move** it there
Inappropriate intimacy	Two roles that each use two or more of the other's features	**Extract** on each side so each asks the other for one thing, then **Move** to break the cycle; or **Inline role** if they're really one role
Divergent change?	Different callers use completely separate parts of a role, so it may be doing two jobs. Parts that work on the same information may still belong together. Calls from the Client aren't counted	**Extract role**, from the finding's Extract a role… button
Data class	Knows things but does nothing with them, or only hands them out (“Get price”)	**Inline role**: fold the role that does the work into it (the finding's Inline … into it… button), or **Move** the work to it
Duplication	The same Knows or Does item on roles of different types	**Move** it into one role, with its copies
Unused	Not traced to the spec and not called in any scenario	Delete the card, or **Inline role** if it holds something worth keeping

Acting on a finding. Some findings carry buttons. A scenario's name opens that scenario on the Sequence tab, where the smell shows up as calls. **Extract a role…** on a Divergent change? finding opens Extract role on that card, with one group of callers' responsibilities already ticked. **Inline … into it…** on a Data class finding appears when exactly one role asks the data class for its data, and opens Inline role with the data class chosen. The previews of Extract role and Inline role end with **Review afterwards**, which says which findings would go and which new ones would appear, before you apply anything.

Shared types. When several roles share a type, Review also looks at their class: together too many responsibilities or dependencies, though no single role has that many, or a cycle that only exists because the roles share the class. These are fixed by changing a role's type on its card, not by refactoring.

Worked examples. The doc *RDD Workbench: Refactoring the Review Smells* has a small example of every smell, with the steps that clear it and the model file to try them on.

Conceptual correlation is at the bottom of the tab. It asks how closely the design's language matches the requirements', using the measure from Jason Gorman's Conceptual tool for code.



- **The first figure** is the share of the design's words that come from the requirements. It's the one Conceptual reports, so you can compare the design with the code later.
- **The second figure** is the share of the requirements' words that are used in the design. A low figure means concepts the spec talks about that no card covers yet.
- **The diagram** puts the requirements' words on the left and the design's on the right; the shared words sit in the overlap. Bigger words are used more. Click a word to see every spec line and card that uses it; card entries link to the card.
- **Role names with no words from the requirements** are listed under the diagram. Ask whether someone who wrote the spec would recognise them.

Design words come from role and type names, Knows and Does items, and role names on associations. Requirements words come from the feature files: feature and scenario names, narratives, steps and table headings, but not example values. Common words are left out, and forms of a word count as one ("reserve", "reserved", "reserves"), but different words don't ("confirm" and "confirmation").

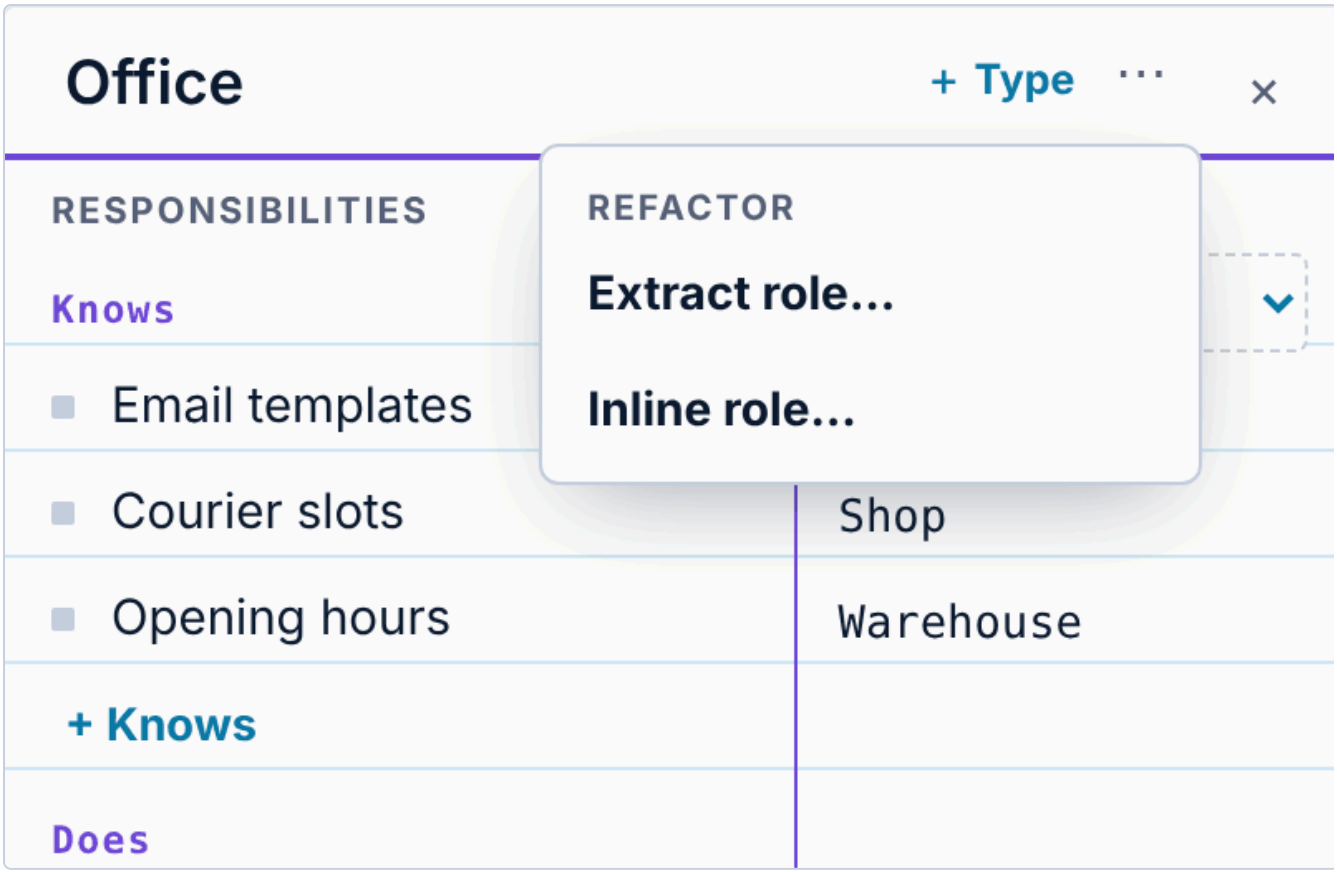
8. Refactor the model

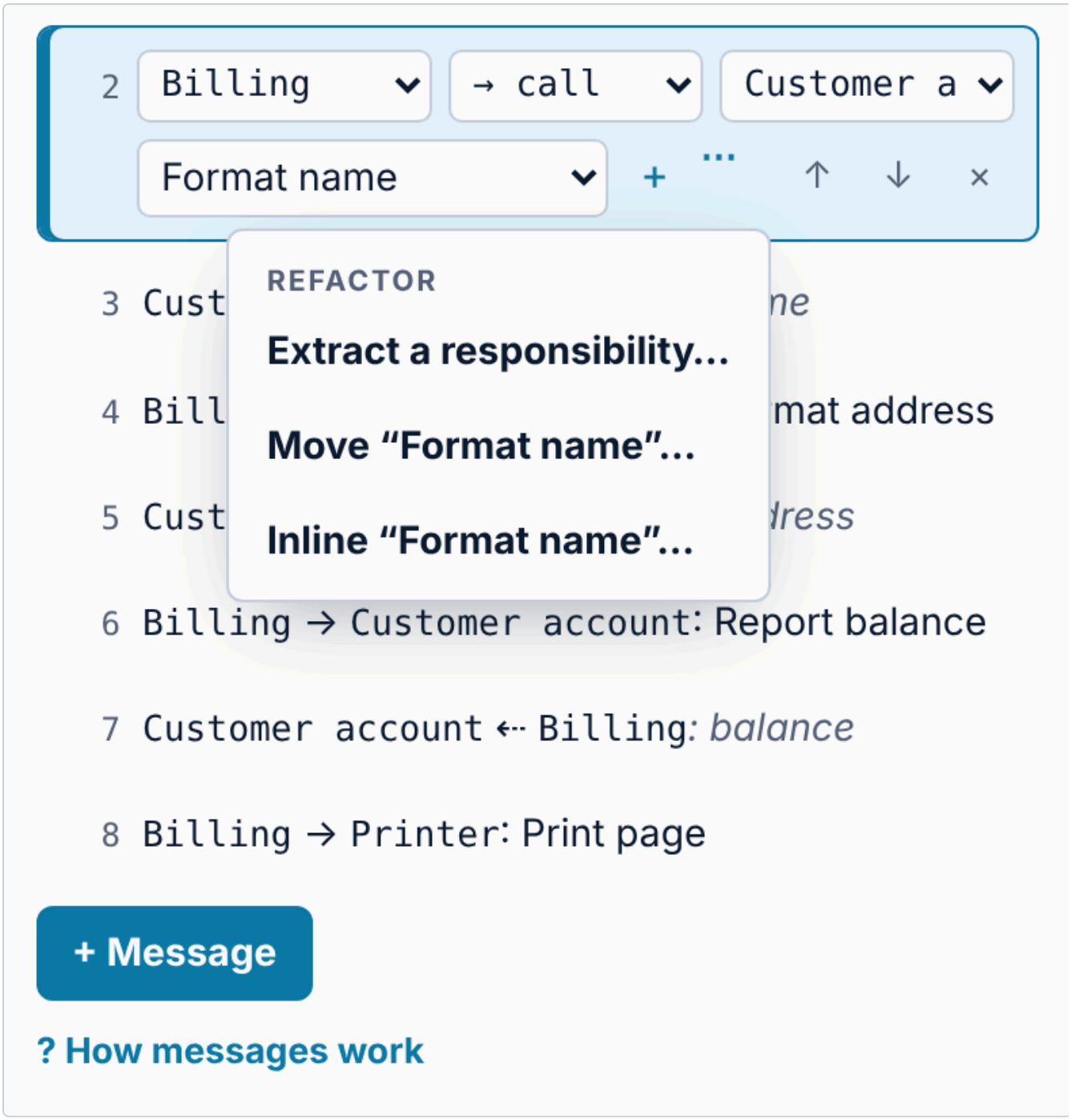
A refactoring changes how the design is put together without changing what it does: every outcome you've realised stays realised, and every outcome that isn't stays that way. That's what makes it safe to act on a Review finding, or simply to try a different shape, in the middle of a workshop.

Every refactoring works the same way:

- **It checks first.** The workbench tries the refactoring on a copy of the model and compares every outcome and entry point before and after. If anything would change, the button stays greyed out and the panel says why (“Can’t move it yet: …”). Nothing is ever half done.
- **It shows a preview** of what will change: responsibilities and Knows items moving, spec links, calls redirected, new collaborations, and ✓ **Behaviour unchanged**.
- **It tidies up as it goes.** The preview lists collaborations no call will use any more and roles left with no messages in a diagram, ticked to be removed in the same step. Untick one to keep it; you're asked about it afterwards instead.
- **It keeps your diagram's style.** In a diagram drawn without return messages, a refactoring adds none, except where a call a role makes to itself needs its return to keep what's nested inside it. The preview says when it draws one.
- **It's one step.** A single Undo puts everything back. Cancel or Esc closes a panel without changing anything.

Refactoring	Where to find it	What it does
Move responsibility	⇥ beside a Does item on a card; ⋯ → Move “…”… on a call	Moves a responsibility to another role or a new one, with its spec links, its calls and the Knows items it needs
Extract responsibility	⋯ → Extract a responsibility… on a call	Turns a run of calls one role makes into one responsibility, done by that role or another
Inline responsibility	⋯ → Inline “…”… on a call	Replaces a call with the calls behind it (the reverse of Extract)
Extract role	⋯ → Extract role… on a card; Extract a role… on a Divergent change? finding	Moves several responsibilities to a new role in one step
Inline role	⋯ → Inline role… on a card; Inline … into it… on a Data class finding	Folds a role into another, and the role goes (the reverse of Extract role)


Office
+ Type
RESPONSIBILITIES
REFACTOR
Extract role...
Inline role...
Knows
Email templates
Courier slots
Shop
Opening hours
Warehouse
+ Knows
Does

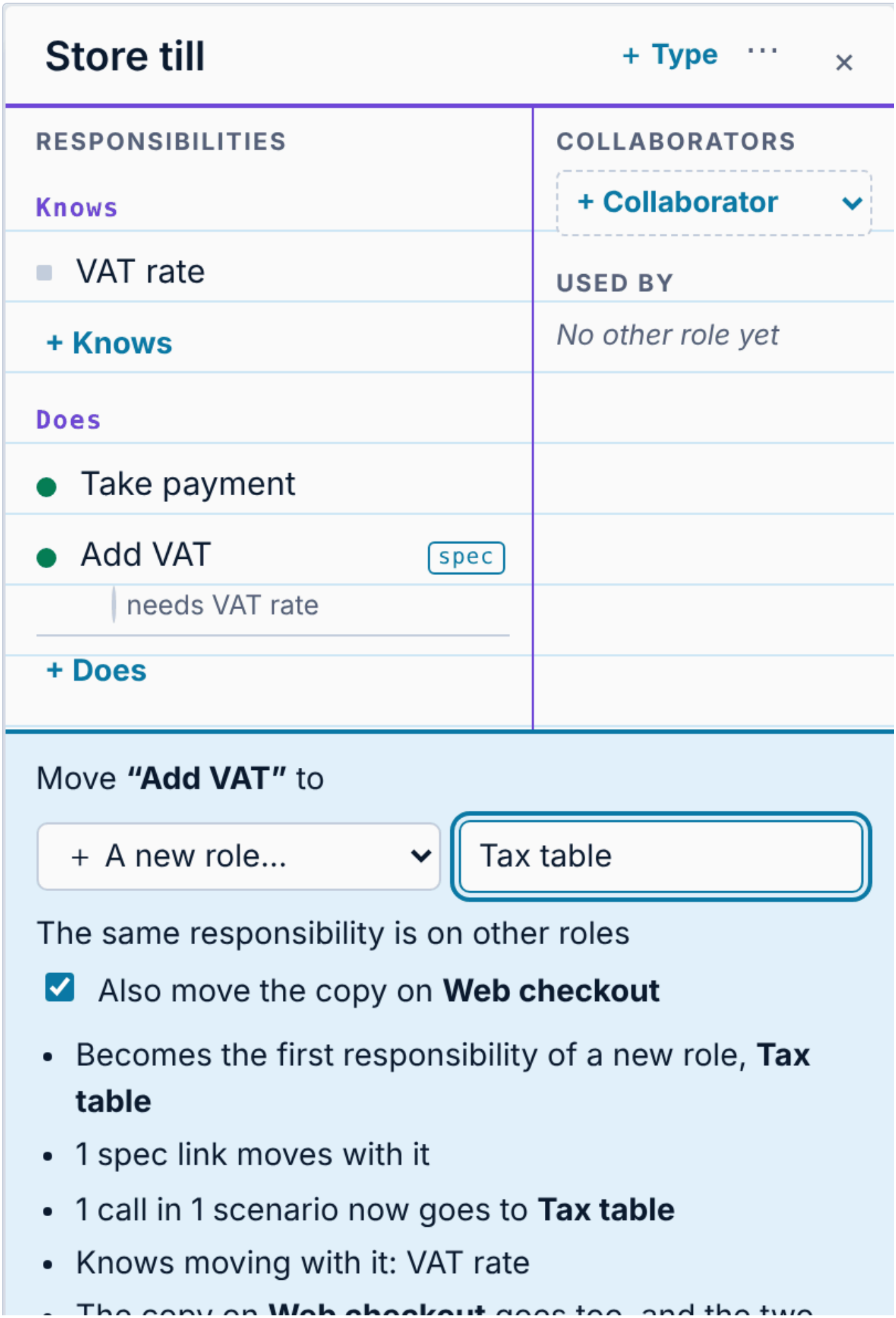


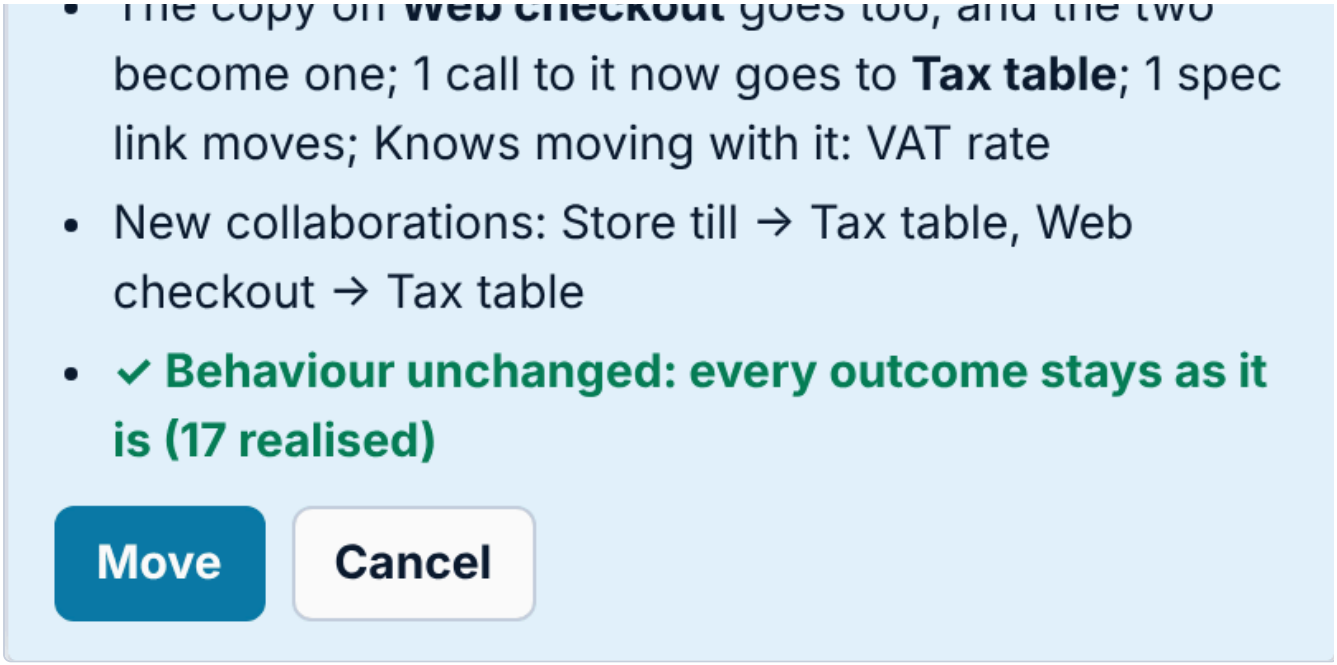
Move a responsibility

Point at a Does item on a card and click ⇥, or open a call to it and choose ⋯ → Move “…”…. Choose the role it goes to, or + **A new role…** and name it. The responsibility leaves the card with its spec links; every call to it now goes to the new role, and the calls the old role made while doing it are now made by the new one. The Knows items it needs go with it, unless other responsibilities on the old card need them too.

- **The same responsibility on other roles.** When roles of another type have the same responsibility (what Review calls Duplication), the panel offers **Also move the copy on …**, ticked. The copies merge into one.

- **What it needs to know.** If the card doesn't record what the responsibility needs, the panel lists the card's Knows items to tick. Ticked items go with it, and are recorded as what it needs.
- A responsibility that's called in a diagram can't move to a role with no name, and a new role can't take an existing role's name.


Store till
+ Type
RESPONSIBILITIES
COLLABORATORS
Knows
+ Collaborator
VAT rate
USED BY
+ Knows
No other role yet
Does
Take payment
Add VAT
spec
needs VAT rate
+ Does
Move “Add VAT” to
+ A new role...
Tax table
The same responsibility is on other roles
Also move the copy on Web checkout
Becomes the first responsibility of a new role, Tax table
1 spec link moves with it
1 call in 1 scenario now goes to Tax table
Knows moving with it: VAT rate



Extract a responsibility

This turns a run of calls into one higher-level responsibility. Open the first call in the run and choose ⋯ → **Extract a responsibility…**. **Include next** takes in the next call the same role makes, with everything it sets off; **Leave out last** takes one away. The run is highlighted in the list and on the diagram. Name what the calls do together, then choose **Who does it?**:

- **The same role** (the default): the role calls itself for the new responsibility, and the calls happen inside that call.
- **Another role, or a new one**: the role calls that role instead, and the calls are now made by it. Use this for Feature envy: open the first envious call, include the others, and give the responsibility to the role they envy.

Loops. A run that cuts across a loop can't be extracted: include the whole loop or none of it. When the run is exactly what a loop repeats, and you give the responsibility to the role being repeated over, the loop moves to the new call. For example, extracting a Basket's *get price* and *get quantity* calls to Item as *subtotal* gives "for each item, Basket asks it for its subtotal". The preview says so. Given to the same role or another one, the loop stays on the calls inside.

The Client's calls (they're entry points), unreachable calls and runs that cut across part of a loop can't be extracted. Extract changes only the scenario you're in.

OUTCOMES 1 of 1 realised

● Then an invoice is printed with the customer's details
✓ Billing: Print invoice

▸ PARTICIPANTS 3 · left to right

MESSAGES

1 Client → Billing: Print invoice

Extract a responsibility from messages 2–7 (3 calls by Billing, highlighted)

Include next | Leave out last

WHAT DO THESE CALLS DO, AS ONE RESPONSIBILITY?

Describe for invoice

WHO DOES IT?

Customer account

- "Describe for invoice" is added to **Customer account**'s card
- **Billing** calls **Customer account** for "Describe for invoice" instead, and the 3 calls Billing made are now made by **Customer account**
- It isn't traced to the spec. The outcomes stay with the calls inside it.
- ✓ **Behaviour unchanged: every outcome stays as it is (17 realised)**

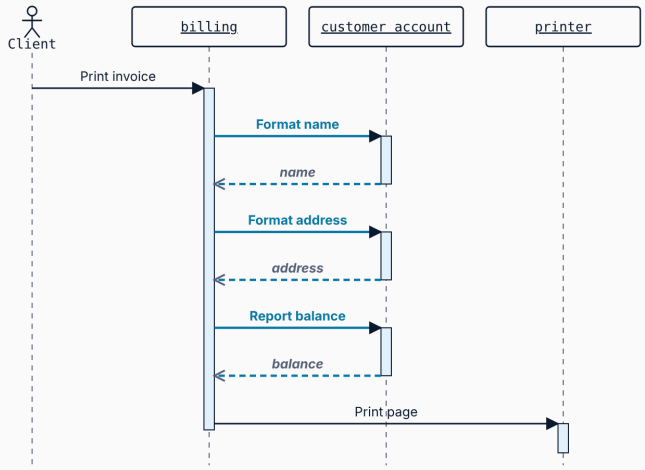
Extract | Cancel

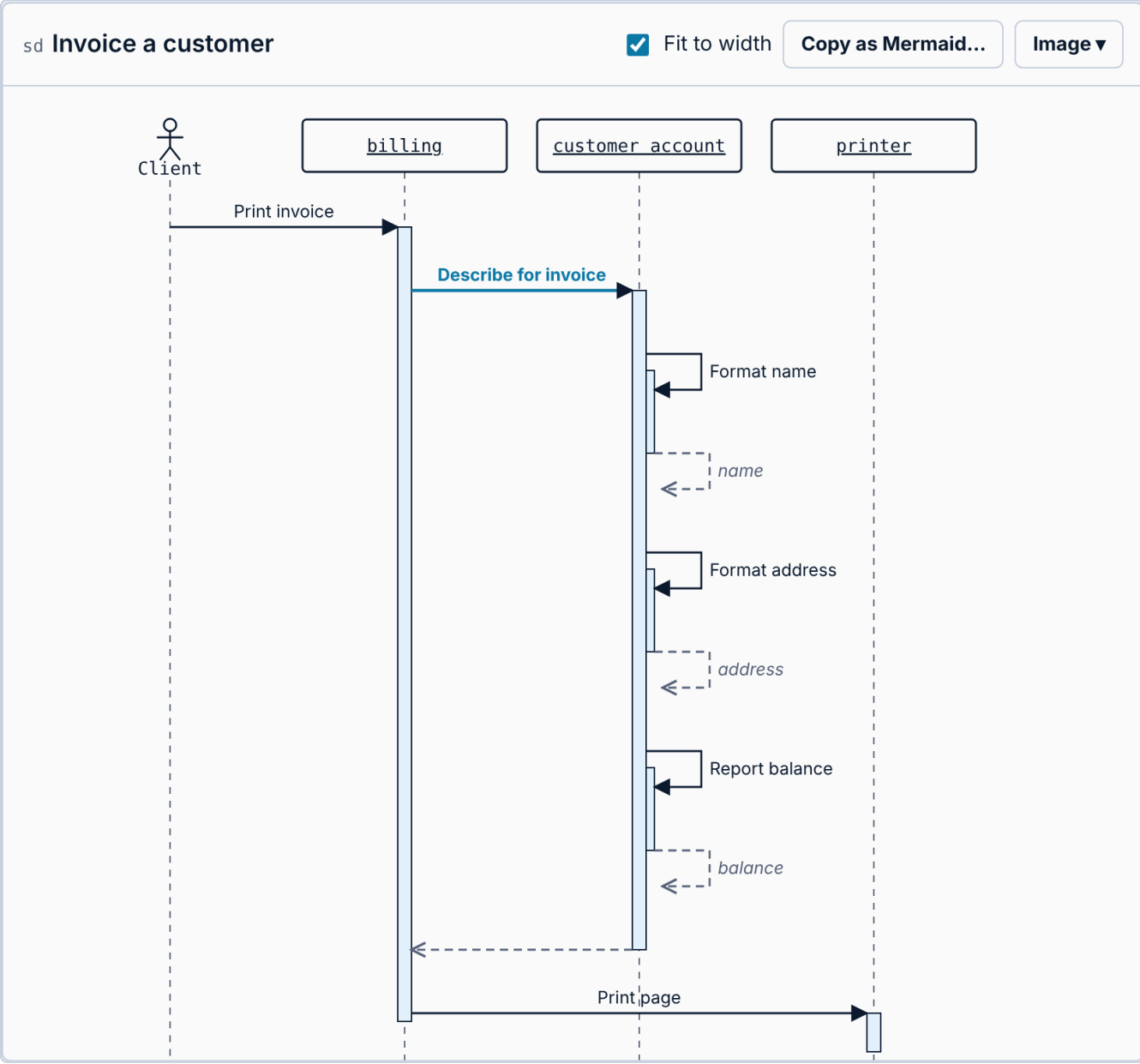
2 Billing → Customer account: Format name

3 Customer account ⇠ Billing: *name*

4 Billing → Customer account: Format address

sd **Invoice a customer** Fit to width | Copy as Mermaid... | Image ▾





Inline a responsibility

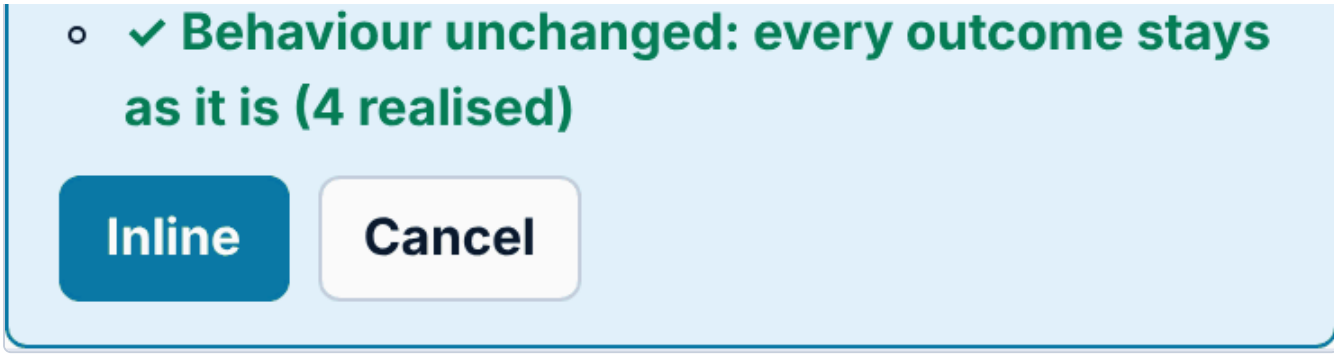
The reverse of Extract: a call is replaced by the calls the receiver made while doing it, now made by the caller. Open the call and choose ⋯ → **Inline “…”…**. Choose **Just this call** or **Every call to it**; when nothing else asks for the responsibility, **Then remove it from …’s card** is ticked. The preview warns when the caller takes over something it doesn't know, or when returned values won't be shown any more.

Inline refuses a call that realises an outcome (inlining it would leave the outcome unrealised) and the Client's calls. To go ahead, assign the outcome to another responsibility in Specs first.

Inline **Checkout**'s "Coordinate placing an order"
(message 2 and what it sets off, highlighted)

- ○ Just this call
- ◉ Every call to it: 2 calls in 2 scenarios
- ☑ Then remove "Coordinate placing an order" from Checkout's card (nothing else asks for it)

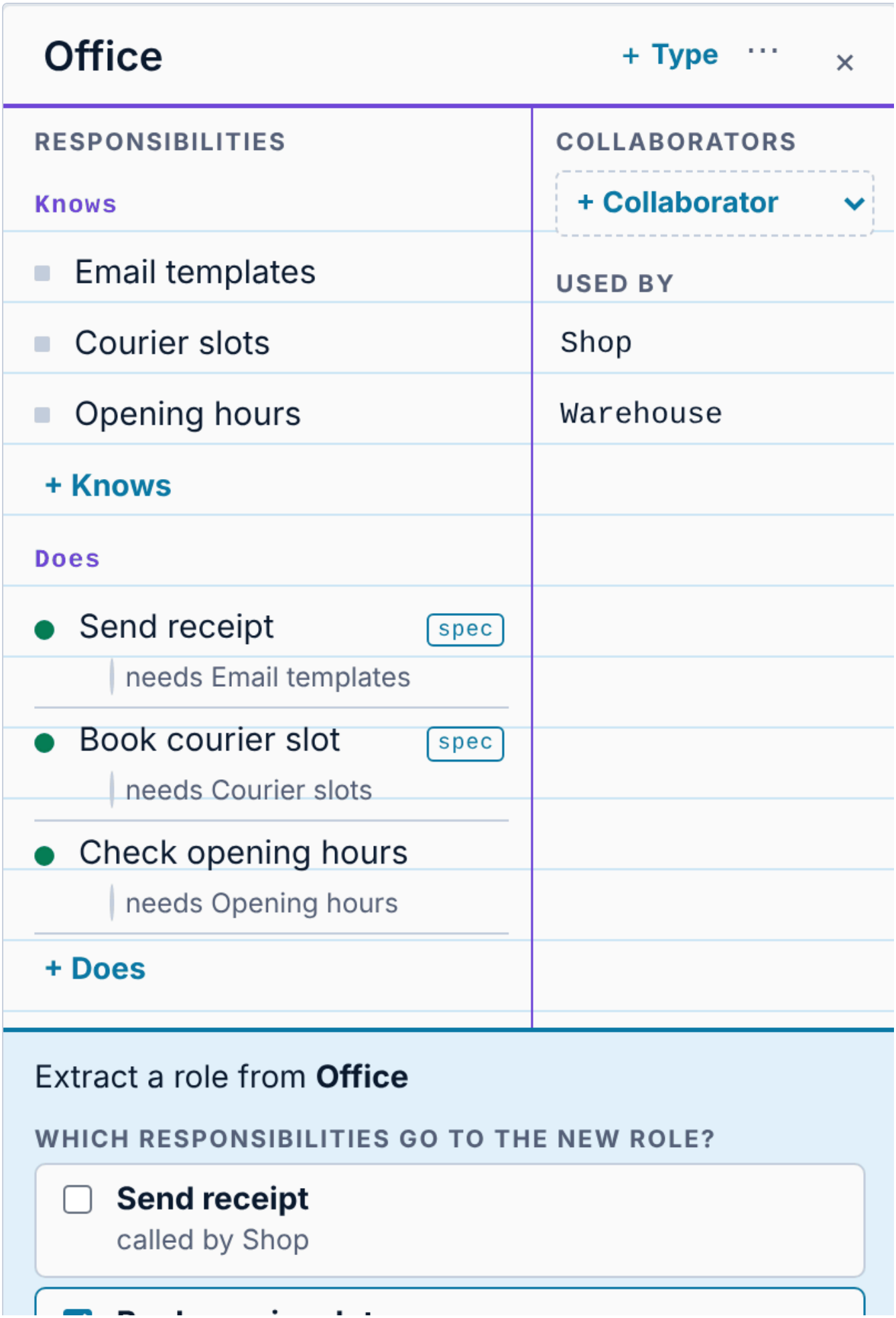
- 2 calls to Checkout for "Coordinate placing an order" in 2 scenarios are replaced by the calls behind them
- The values it returned ("confirmation", "out-of-stock notice") aren't shown any more.
- New collaborations: Customer → Basket, Customer → Pricing engine, Customer → Inventory, Customer → Payment gateway, Customer → Order
- ☑ Delete the collaborations no call uses any more: Customer → Checkout, Checkout → Basket, Checkout → Pricing engine, Checkout → Inventory, Checkout → Payment gateway, Checkout → Order
- ☑ Remove what's left with no messages from the diagrams: **Checkout** in "Successful order", "Item out of stock"

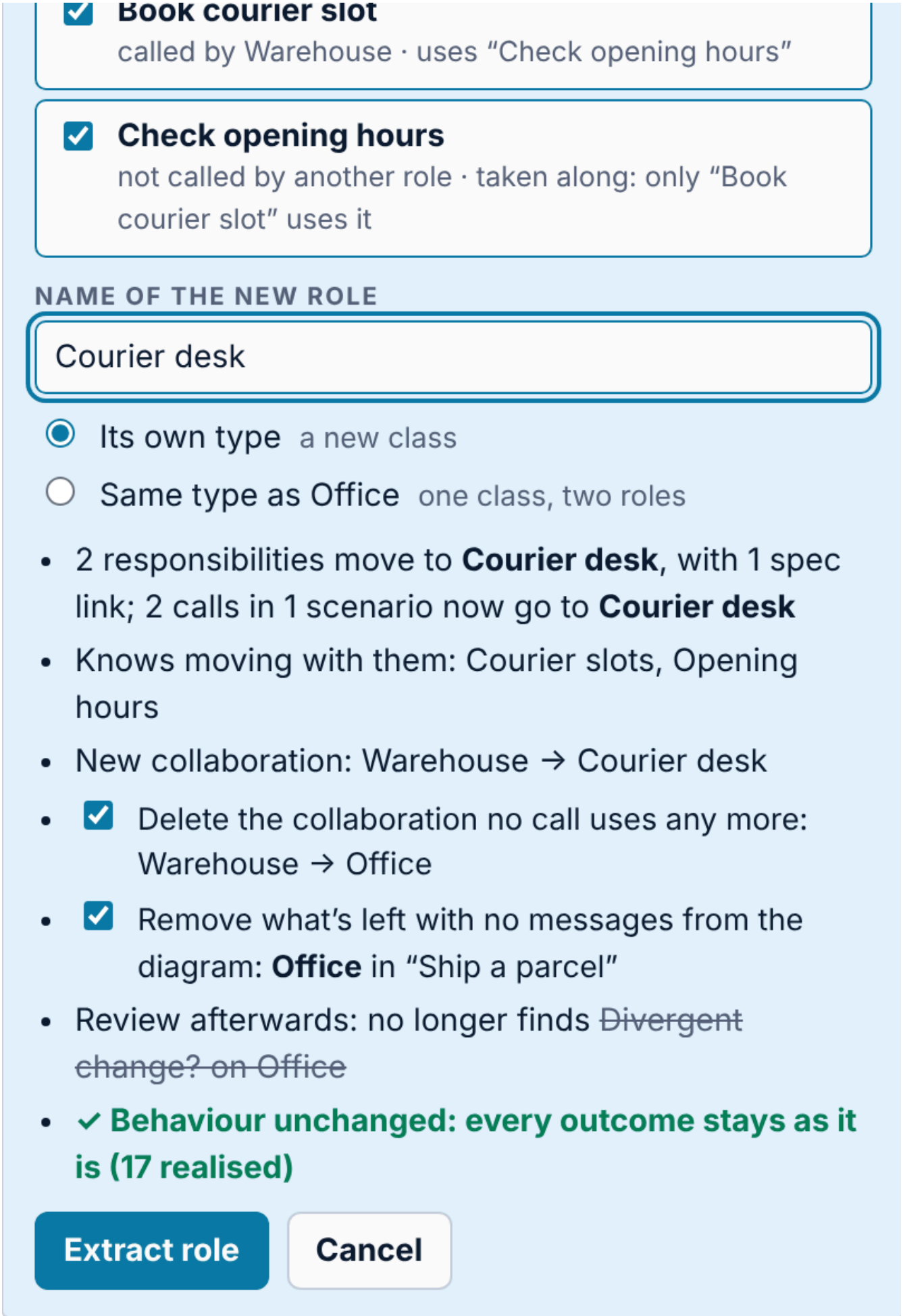


Extract role

Several Moves into a new role, done as one. Choose ⋯ → **Extract role…** on the card (or **Extract a role…** on a Divergent change? finding), tick the responsibilities that go, and name the new role.

- **Dependencies come along.** A responsibility only the moving part uses is ticked for you (“taken along”). One that both parts use stays, and the new role calls the old one for it (“Kept as a call”); if the two would call each other, the preview warns of a cyclic dependency.
- **What they need to know.** Knows items move with the responsibilities that need them. Knows both parts need stay, with a warning that the two jobs may belong together after all. If needs aren't recorded, tick the Knows items that should go.
- **Type.** The new role is its own type (a new class) unless you choose **Same type as …** (one class, two roles).
- **Review afterwards** says which findings would go, and any new ones.

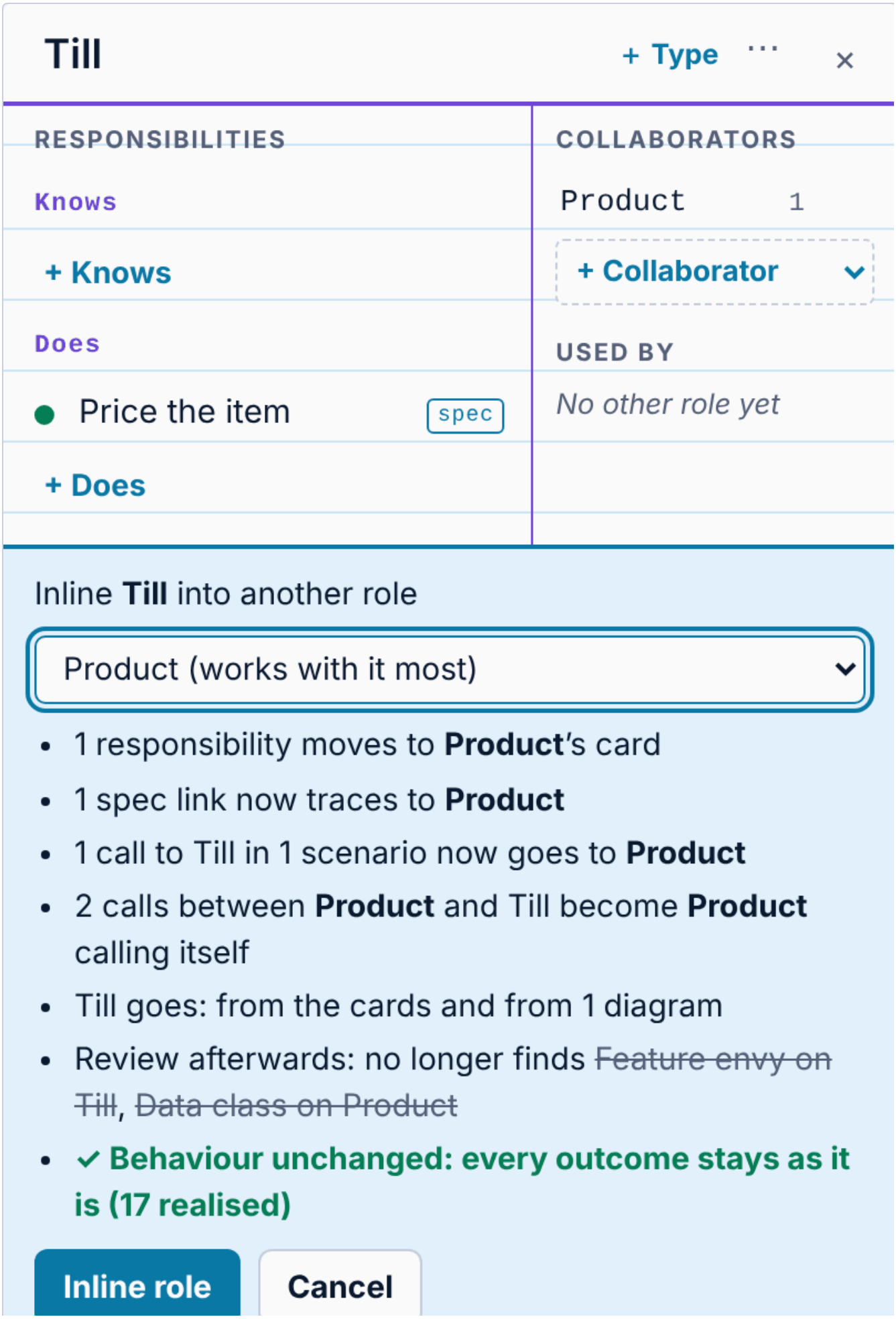

Office
+ Type
RESPONSIBILITIES
Knows
Email templates
Courier slots
Opening hours
+ Knows
Does
Send receipt
spec
needs Email templates
Book courier slot
spec
needs Courier slots
Check opening hours
needs Opening hours
+ Does
COLLABORATORS
+ Collaborator
USED BY
Shop
Warehouse
Extract a role from Office
WHICH RESPONSIBILITIES GO TO THE NEW ROLE?
Send receipt
called by Shop


Book courier slot
called by Warehouse · uses “Check opening hours”
Check opening hours
not called by another role · taken along: only “Book courier slot” uses it
NAME OF THE NEW ROLE
Courier desk
Its own type a new class
Same type as Office one class, two roles
2 responsibilities move to Courier desk, with 1 spec link; 2 calls in 1 scenario now go to Courier desk
Knows moving with them: Courier slots, Opening hours
New collaboration: Warehouse → Courier desk
Delete the collaboration no call uses any more: Warehouse → Office
Remove what’s left with no messages from the diagram: Office in “Ship a parcel”
Review afterwards: no longer finds ~~Divergent change? on Office~~
✓ Behaviour unchanged: every outcome stays as it is (17 realised)
Extract role
Cancel

Inline role

The reverse of Extract role: fold a role into another, and the role goes. Choose ⋯ → **Inline role…** on its card (or **Inline … into it…** on a Data class finding) and the role to fold it into; the one it works with most is suggested. Its Knows and Does items, spec links and collaborations go to that role, and items with the same wording become one. Calls to it go to that role, its own calls are made by that role, and calls between the two become the role calling itself, drawn nested where their returns are drawn. **Review afterwards** warns if the combined role now has too many responsibilities.

Inline role refuses when something one role does has the same name as something the other knows, or when some calls between the two have returns drawn and others don't, which would leave it unclear where each self-call ends. Draw the missing returns first.


Till
+ Type
RESPONSIBILITIES
COLLABORATORS
Knows
Product 1
+ Knows
+ Collaborator
Does
USED BY
Price the item
spec
No other role yet
+ Does
Inline Till into another role
Product (works with it most)
1 responsibility moves to Product's card
1 spec link now traces to Product
1 call to Till in 1 scenario now goes to Product
2 calls between Product and Till become Product calling itself
Till goes: from the cards and from 1 diagram
Review afterwards: no longer finds ~~Feature envy on Till~~, ~~Data class on Product~~
✓ Behaviour unchanged: every outcome stays as it is (17 realised)
Inline role
Cancel

What refactorings don't do

- **Calls that aren't next to each other.** Extract takes calls one role makes in a row. Move a call out of the way with the arrows first; that's an ordinary edit, so check the outcomes afterwards.
- **Other scenarios.** Extract changes only the scenario you're in.
- **Re-pointing outcomes.** Assigning an outcome to a different responsibility is a deliberate edit in Specs, not part of a refactoring.
- **Shared types.** Smells on a class that several roles share are fixed by changing a role's type.

Undo and redo

↶ and ↷ at the top right undo and redo changes to the model, on any tab. So do Ctrl+Z (⌘Z on a Mac), and Ctrl+Shift+Z (⌘⇧Z) or Ctrl+Y, except while you're typing in a text box, where they undo your typing as usual.

- **One action is one step.** Typing a name is one step, even with pauses. A refactoring, with its tidying up, is one step. Changing a call's receiver and then adding or moving the responsibility it asks for is one step.
- **Undo takes you there.** If the change was made on another tab or scenario, Undo goes back to it and says so (“Undone, on the CRC cards tab”).
- **Undo history lasts until you reload the page.** Your model itself is saved as you work.

Saving and sharing your model

Your work saves automatically in the browser you're using. It won't follow you to another browser or computer, and clearing your browser data removes it, so download a copy at the end of each session. Click **Model ▾** at the top right for these options:

- **New** starts a blank model.
- **Save As…** saves the whole model as a .json file, named after your first feature and today's date.
- **Open…** picks up a model you saved earlier, or one someone has sent you.
- **Example Model** loads the worked example.

New, Open and Example Model replace the model you have loaded, so each one asks first: “This will overwrite the currently loaded model. Proceed?” Choose **Yes** or **Cancel**. If you replace it by mistake, **Undo** brings it back. Models saved from the earlier, roles-first workbench open here unchanged. Models saved here also record what each responsibility needs to know; the earlier workbench ignores that part.

Quick reference

Marker	Where	Meaning
Number on 2 · Name the roles	Specs	Roles still waiting for a name
Empty / blue / green circle	Specs	Outcome not yet assigned / assigned / realised in its scenario's sequence diagram
Under Suggested, *knows Price* in Which role?	Specs	That role already knows something the responsibility needs
already does this in Which role?	Specs	That role already has this responsibility
Italic chip with a dashed edge in the Roles tray	Specs	A role with no name yet
Green underline on a word	Specs	Added to what a role knows
Shaded word	Specs, Name the roles	Names a role
Darker shading	Specs, Name the roles	A phrase linked to a role
Blue background on a word	Specs	Mentions the role you clicked in the tray
Diamond by a When step	Specs	A request from the Client; filled once assigned. Not counted in progress
+2 on a chip	Specs	The same responsibility is linked to 2 other outcomes
✓ on a chip	Specs	That assignment is realised in the scenario's sequence diagram
NOT on a chip	Specs	Negative outcome: realised when the responsibility is not called
Orange chip with Re-add to card	Specs	The linked item was removed from its card
Name it later and a dashed line under the title	CRC cards	A role with no name yet
needs … under a Does item	CRC cards	What that responsibility needs to know
spec / spec ×3 badge	CRC cards	Linked to one / three outcomes; hover to see which
Filled spec badge	CRC cards	Linked from two or more features
entry badge	CRC cards	Called by the Client: assigned from a When step
Green / hollow dot by a Does item	CRC cards	Called / not yet called in a sequence diagram
Dotted collaborator name	CRC cards	Not used by any call in a sequence diagram
Grey 1 after a collaborator	CRC cards	Multiplicity not set, so it's 1
⇥ beside a Does item	CRC cards	Move it to another role

Marker	Where	Meaning
⋯ in a card's header	CRC cards	Refactor: Extract role, Inline role
Name its role to realise this scenario	Sequence	The outcome's role needs a name before it can take part
Dotted orange message	Sequence	Unreachable: its sender hasn't been called earlier in the scenario
Not on …'s card	Sequence	The receiver doesn't have that responsibility: add it, or move it here
⇠ on a message	Sequence	That call has no return yet; click to add one
↻ on a message	Sequence	The receiver is a collaborator the sender has many of; click to repeat the call
Frame tagged loop	Sequence	The messages inside repeat, under the condition in brackets
Second bar on a lifeline	Sequence	A role calling itself, with the return drawn
⋯ on an open call	Sequence	Refactor: Extract a responsibility, Move, Inline
Add a getter in a call's responsibility list	Sequence	"get" + each thing the receiver knows; choosing one adds it to its card and uses it
Underlined name on a lifeline	Sequence	The role taking part (": Person" adds its type)
Grey operation	Class diagram	Not called in any sequence diagram yet
Number at an arrow's end	Class diagram	Multiplicity: 1 unless set
Orange line under the diagram	Class diagram	Roles left out until they're named
Orange smell label	Review	A possible smell; what usually helps is in section 7
Blue / green / orange words	Review	Requirements only / shared / design only
✓ Behaviour unchanged	Refactoring panels	Every outcome stays as it is, so the refactoring can go ahead
Prompt at the bottom of the screen	Any tab	A collaboration or participant nothing uses any more: delete it or keep it
Demo button; Demo: nothing is saved	Header	Watch the demo; while it runs, your own model is put aside and nothing is saved

Common problems

- **I can't add a role to a sequence diagram.** It hasn't been named yet. Name it in 2 · Name the roles, or click its title on the CRC cards tab.
- **A role is missing from the class diagram.** Same reason: only named roles are drawn. The line under the diagram links to the ones waiting.
- **Several cards look like the same role.** Fold one into the other with ⋯ → **Inline role…**. Or merge their items with ⤵ and delete the empty card.

- **An assignment is flagged after I edited a step.** Links follow the step's exact wording. Remove the flagged link and assign the step again.
- **A message says “Not on …’s card”, or its label turns orange.** The receiving role doesn't have that responsibility. Add it to the card, move it there from the role that has it, or pick one of the receiver's responsibilities.
- **A collaborator appears that I didn't intend.** A call that asks for a responsibility makes the receiver a collaborator of the sender once you close the message, so check the call's sender before you move on. If you correct it later, the workbench asks whether to delete the collaboration nothing uses any more.
- **There's no ↻ to repeat a call.** Set the collaborator's multiplicity on the sender's card to * or 1..*.
- **A refactoring button is greyed out.** The panel says why. Usually an outcome would change: to inline a call that realises an outcome, assign the outcome to another responsibility first; to extract calls inside a loop, include the whole loop or none of it.
- **Undo did more than I expected.** A refactoring and its tidying up are one step, and so are a change of receiver and the card fix after it. Redo puts it back.
- **My work has gone.** It was saved in a different browser, or browser data was cleared. Open your last saved copy with Model ▾ → Open….
- **The demo's panel covers what I want to see.** Drag it by its blue header; it stays where you put it until the demo ends.

bardd 1.0.8 · A Codemanship application · www.codemanship.co.uk · © Codemanship Ltd 2026