PFV Practitioner Guide

Map it. Diagnose why it waits. Act on evidence. Seven steps, one process, about an hour end to end. The method is not domain-specific — it works the same way on a project, a recovery, an access request, a migration or a vulnerability. Only the cost of the waiting changes.

Start in the elevator

Two questions, fifteen seconds, no software. **How many items are open right now? How many finish in a week?** Divide the first by the second — that is how long each one really takes. If that number is acceptable, stop here and go and do something else. If it is not, the rest of this page is how you find out why.

The engagement, in six stages

Triggered by an ugly outage, a new leader, a budget challenge or a new technology. The whole sequence runs on one process at a time.

1	**Start** how many items are open, how many finish in a period. Divide. Does that cycle time work for the business? If it does, stop.
2	**Map** one recurring process. Per step: the task, who performs it, hands-on work, waiting, whether that time is owned by an external team, and where work is sent back to an earlier step.
3	**See** it as one timeline drawn to scale — green for work, red for waiting you control, yellow for work or waiting owned outside your organisation.
4	**Classify** the waits — internal, external, required, or rework related.
5	**Diagnose** where the largest controllable exposure sits, and why the work waits there.
6	**Improve** one thing. Then measure the same process again.

Classifying the waits is what makes the diagnosis actionable. Internal waits are yours to remove, external ones are not, required ones are controls doing their job, and rework is the only one that returns capacity. Step 6 is a small separate scope, and the only way anyone learns whether the recommendation landed.

Step 1 — Choose one process

One recurring process with a clear start and a clear finish. Something that has run enough times to have a normal shape. Bring the people who do the work: the queue times are the point of the exercise, and they are the numbers managers get most wrong.

Step 2 — Map it, and record two numbers per step

Cut a new step where custody changes hands, where a queue forms, and where the clock owner changes. Twenty steps is the ceiling; most useful maps are eight to fourteen. For each step record **hands-on hours for one item** and the **waiting time that step consumes**. Estimate if you must, and label the estimate.

Waiting is elapsed time the step consumes that is not hands-on work, recorded on the step whose completion it delays. If an item sits four days before anyone triages it, that is four days on *triage* — never a row called “waiting”. A queue is not a step: give one a step number and you inflate the count, leave “who does it?” unanswerable, and hide which real step the delay is holding up. Three causes sit inside that one number:

Cause	What happens	What it usually means
Entry queue	The item is ready. The person or system that must act is not.	Capacity, priority order, or batching at the receiving end.
Handoff gap	The previous step finished and nobody moved it on. It sat in an outbox.	An unowned transition. Often the cheapest wait in the map to remove — and still recorded on the step it delays.
Mid-step block	Work *started*, then stalled — a question went unanswered, an input did not arrive — then resumed.	A dependency inside execution. The step is occupied for the whole span even though little is happening.

You do not classify these while mapping. Asking somebody to would slow the conversation for no gain, and the **“waits on”** field already carries it: waiting on another team is usually an entry queue, waiting on nothing at all is usually

a handoff gap, and waiting on data or a system mid-execution is usually a block. The six questions stay six.

> It matters for scheduling. A step that waits to *start* leaves its resource free, so other work can fill the gap. A step that starts and stalls occupies its resource for the whole span, and nothing else can run in it.
>
> Getting the cuts wrong is the commonest way a first map disappoints — too fine and it is unreadable, too coarse and the queues hide inside the steps.

Typing the steps is the best way to learn the method, and it is not the only way to get a process in. Two blank templates download from the file panel above the step grid, and either layout is accepted — **one row per step**, which is what a ticket export gives you, or **one column per step**, which is how the worksheet exports. You do not have to choose correctly: the loader works out which way round a file is and turns it if needed. Comma, tab and semicolon files all work, as do quoted fields, European decimal commas, and values written as *3h*, *2.5 days* or *20%*.

	Column	Why
Required	Step	What happens, in a few words.
Required	Wait (days)	Elapsed time completing the step that is not hands-on work. Without it there is nothing to see.
Recommended	Work (hours)	Hands-on time for one item, not elapsed time.
Recommended	Waits on	What prevents completion, in the words the customer used — another team, an approval, information, a system, an external partner, capacity, a window. Drives the improvement advice.
Optional	Performed by	Person, role, team or system. Blank is allowed, and may itself be a finding.
Optional	Sent back (%)	Share of items that return to an earlier step and repeat work. Without it you lose the rework analysis — the only route to extra capacity.

A missing optional column loads as zero and raises a warning; the file still goes in. Only a file with no numbers anywhere is refused, and it says why. **Keep the column headings exactly as the template writes them** — those labels are how the tool knows which field is which.

Save this process writes what is on screen to a CSV on your own machine, and it loads straight back in next month to measure again. Nothing is uploaded and the site stores nothing — the only copy of your data is the file you saved.

Step 3 — See where, and only where

The largest queues tell you where to investigate. **Resist the reflex to fix anything yet.** The biggest wait does not tell you why it exists, and it does not prove you found a bottleneck. Note the top one or two waits; everything that follows carries them forward, so every finding traces back to the map.

Step 4 — Diagnose the wait

Six mechanisms. Work down them in order: it rules out the cheap-to-confirm, non-capacity explanations first — the ones that do not end in “we need more people.”

Mechanism	What it is	Evidence that confirms it	Watch out
Capacity constraint	Demand persistently meets or exceeds what the step can complete.	Work accumulates before the step; the resource is highly utilised; arrivals ≈ completions; the queue **grows over time**.	Concentration is not capacity. The longest wait is not automatically a bottleneck.
Batching / schedule	Work is held for a meeting, review cycle, window or minimum batch size.	Items wait **while capacity is free**; completions cluster around the cycle.	The most common false capacity signal. An idle approver with a long queue is almost always batching.
Dependency / blocking	Work cannot proceed until another team, vendor, input or decision arrives.	Blocked status; external ownership of the next action; age tracks *their* response time.	Adding capacity here does nothing — the step was never the limiter.
WIP overload	Too many items active at once; multitasking ages everything.	High active-item count per person; many started, few finished; frequent expedites.	Staffing can look adequate while the queue grows. Attention is the scarce resource.

Rework loop	Work repeats because inputs were incomplete or defects surfaced late.	Items move backward; reopenings and resubmissions; high rejection rate.	Rework consumes real capacity and reads as “not enough people.”
Policy / control	The wait is required by design — approval, separation of duties, compliance.	The delay is intentional; the control owner can name its risk purpose.	Not automatically waste. Ask whether the objective can be met in less elapsed time.

A seventh verdict is legitimate: unknown, needs evidence. When the indicators do not support a classification, go and look before prescribing anything.

Common false diagnoses

Longest wait = bottleneck	Length locates; it does not explain.
Idle-approver batching read as capacity	A long queue with a free resource is almost never capacity.
Rework read as understaffing	A busy step may be redoing work rather than doing new work.
Policy delay treated as pure waste	Ask what the control is for before you cut it.
Forcing a verdict	“Unknown, collect evidence” beats a confident wrong answer.

Step 5 — Say what removing the wait would return

Return	When it is real
Capacity	Only when effort is **eliminated** — work stops being done twice. Hands-on hours multiplied by the send-back rate is the size of it. **The only one that reduces what the process costs to run.**
Flow	Whenever the wait is on the path. Each item finishes sooner — which is not the same as more items finishing.
Exposure	Always. The undesirable condition stays open for less time, and for many processes that *is* the product.

The question that separates them: if we shorten this wait, does it merely finish this item sooner — or does it let the system complete more items? Settle it by counting arrivals against completions for four weeks. A growing queue means completions are capped and shortening the wait raises output; a stable queue means you are buying speed and exposure reduction, not capacity.

Step 6 — Verify before you commit budget

Pull the flagged steps from the status-transition logs your systems already record — issue changelogs, task audit tables, case timelines, ticket histories. They carry both the times and the mechanism. **Verification is what turns a self-reported diagnosis into a measured one**, and it is the difference between a number that survives a budget review and one that does not.

> A mechanism you have named but not evidenced is a **hypothesis**. Two or more indicators make it **evidence-supported, self-reported** — deliberately not *verified*, because nobody has read your logs.

Step 7 — Fix one thing, then measure again

Write down what you expect to change, and by roughly how much, **before** you change anything. Then run one small reversible experiment. If the number moves as predicted, the mechanism held. If it does not, the mechanism was something else — return to step 4 rather than trying harder at the same fix.

Re-measure with the same map. **The pile moves:** fix the biggest wait and the next one is usually somewhere you did not expect. If total lead time did not fall, the step you fixed was never the constraint.

Where this stops. PFV decomposes flow, and not all delay is flow. If nothing was looking — no rule fired, no ticket was ever raised — there is no queue to decompose and the map will mislead you. Being clear about where the method stops is what makes the rest of it worth trusting.