Diagnosing why work waits

Finding the biggest wait is the easy half. This is the other half: working out **why** it is there, before spending anything on it. Six mechanisms, the evidence that confirms each, a decision path, and the five wrong answers people reach for most often.

Why this matters more than the number

A map tells you **where** the time sits. It does not tell you why, and the two most expensive mistakes in process work both come from skipping the question. The first is treating the longest wait as the bottleneck. **Length locates; it does not explain.** The second is reading every long queue as a shortage of people, which is the conclusion that costs the most and is right the least often.

> Diagnose before you prescribe. If you cannot name the mechanism, the honest answer is that you do not yet know.

First, what counts as waiting

Waiting is elapsed time a step consumes that is not hands-on work, and it is recorded on **the step whose completion it delays**. If an item sits four days before anyone triages it, that is four days on *triage* — not a row called "waiting". PFV follows the item through the process, not the person's calendar, so waiting is never a step of its own.

Within that single number there are three causes. They differ in **why** the item is waiting, not in where the wait is written down:

Cause	What happens	What it points at
Entry queue	The item is ready; the person or system that must act is not.	Capacity, priority order, or batching at the acting step.
Handoff gap	The previous step finished and nobody moved the item on — an outbox, an unread notification.	An unowned transition. Often the cheapest wait in a map to remove, and the one most often missed.
Mid-step block	Work started, then stalled on a question or a missing input, then resumed.	A dependency the step cannot satisfy while running.

You do not have to classify these while mapping, and asking somebody to would slow the conversation for no gain. 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.

> One place it changes the arithmetic. A step waiting to *start* leaves its resource free, so other work can run in the gap and the step can honestly be marked parallel. A step that starts and then stalls occupies its resource for the whole span.

The six mechanisms

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 wait tracks arrival timing within it.	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; queue age tracks *their* response time, not this step's capacity.	Adding capacity here does nothing — the step was never the limiter.
WIP overload / switching	Too many items are active at once; multitasking ages everything.	High active-item count per person; many started, few finished; frequent expedites; ageing work while new work keeps starting.	Staffing can look adequate while the queue grows. Attention, not headcount, is the scarce resource.

Rework loop	Work repeats because inputs were incomplete or defects surfaced late.	Items move backward; reopenings and resubmissions; high rejection or clarification 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; removing it would change a control objective.	Not automatically waste. Ask whether the objective can be met in less elapsed time.

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

A decision path

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

Held for a schedule or window?	→ **Batching**
Blocked on an external prerequisite or team?	→ **Dependency**
Work piling up before a highly-utilised step?	→ test for **Capacity**
Many items active at once, or constant interruption?	→ **WIP overload**
Work repeats or moves backward?	→ **Rework loop**
The wait is required by design?	→ **Policy / control**
None clearly apply?	→ **Unknown** — collect evidence before prescribing

The five false diagnoses

Longest wait = bottleneck	Length locates; it does not explain. Test it before you act — arrivals against completions, whether the next step ever starves, and whether the resource is busy on *this* work.
Idle-approver batching read as capacity	A long queue with a free resource is almost never capacity. Look at whether anyone is actually busy before concluding anything about headcount.
Rework read as understaffing	A busy step may be redoing work rather than doing new work. Hiring into it buys you more of the same rework.
Policy delay treated as pure waste	Ask what the control is for before you cut it. The right question is whether its objective can be met in less elapsed time, not whether the wait can be deleted.
Forcing a verdict	“Unknown, collect evidence” beats a confident wrong answer. Every other mistake on this list starts with someone unwilling to say it.

Say what you actually know

A mechanism you have named but not evidenced is a **hypothesis**. Two or more evidence indicators make it **evidence-supported**, self-reported — deliberately not *verified*, because nobody has read your logs. **Verified** means you pulled it from status-transition data: Jira issue changelogs, ServiceNow task audit tables, SOAR case timelines. Those already exist in systems you run, and they carry both the times and the mechanism.

> Check only what you have genuinely observed. A number whose origins are attached survives a meeting that a confident number will not.

Then test it, cheaply

Treat the diagnosis as a hypothesis with a prediction attached. Write down what you expect to move and by roughly how much **before** you change anything, then run one small reversible experiment: expedite five items past the step, raise the review cadence for a fortnight, cap work in progress for a month.

If the number moves in the predicted direction, the mechanism held. **If it does not, the mechanism was something else** — return to the decision path rather than trying harder at the same fix. That negative result cost you two weeks and saved you a budget cycle.

And re-measure after every change. 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.

What removing a wait actually returns

Naming the mechanism tells you **how** to remove a wait. It does not tell you **what you get**, and those are three different things that get carelessly treated as one. Report them separately or you will promise the wrong one.

Return	What it means	When it is real
Capacity return	Fewer resource hours per outcome. Effort is **eliminated** — rework, duplicate processing, unnecessary coordination.	When work stops being done twice. Multiply hands-on hours by the send-back rate: that is the capacity coming back. **The only one of the three that reduces what the process costs you.**
Flow improvement	Outcomes move through faster. Elapsed time falls.	Whenever the wait is on the path. Always true, and always worth less than it looks — see the question below.
Risk / exposure reduction	The undesirable condition stays open for less time.	Always, and it needs no further justification. A vulnerability open 11 days instead of 78 is a materially different risk at identical volume.

The question that decides it

If we shorten this wait, does it merely finish this item sooner — or does it let the system complete more items? Everything else on this page is preparation for answering that, and most process work never asks it.

If…	Then…
Effort is eliminated — rework stops, a duplicate step goes	Partly **more**. Those hours come back whatever else is true, because they simply stop being spent. Capacity return is unconditional.
No effort is eliminated — the item just waits less	**Sooner**, unless the system is flow-limited. It becomes *more* only if completions are currently capped by work in progress, available slots, or a dependency this wait is holding.
The wait is external — a vendor, a customer, another organisation	**Neither**, on its own. You cannot shorten it. The only question worth asking is what you can start in parallel while it runs.

How to settle it in four weeks: count arrivals against completions at that step. If the queue is growing, completions are capped and shortening the wait raises output. If it is stable, you are buying speed and exposure reduction — both worth having, neither of them capacity. That is the same arrival-versus-completion check that opens the bottleneck test, and it answers two questions at once.

> The discipline this imposes is worth the trouble. “We will save 40 days a year” is a claim about flow. “We will free up 300 hours” is a claim about capacity. They are not the same sentence, and only one of them survives somebody checking.

When to reach for each of the three tools

The rubric, the model and the worked examples do different jobs, and using the wrong one is how people end up with a plausible answer to a question they were not asking. Each of them starts where this page ends — with a wait you have located and a mechanism you have named.

Tool	Use it when	What it gives back	Do not use it to
The Rubric dewood.org → Rubric	You have named a mechanism and want to know **what to actually do** about it. Also when you are not sure the wait is a bottleneck at all — the four checks live here.	The mechanisms that act on that shape of wait, ranked, with what doing each one looks like at your maturity level, worked examples, and how you will know it worked.	Decide whether the fix is worth funding. It ranks; it does not size.
The Model dewood.org → Model	You have chosen a fix and are about to spend time or money on it. Use it **before** you act, never after.	A predicted number of days removed with a 90% band, committed to a dated ledger, and a score once you enter what actually happened.	Justify a decision already made. A prediction written afterwards is a story.
How to Improve dewood.org → How to Improve	You want to know what this kind of change is typically worth, or you are arguing with somebody about whether it is worth doing at all.	Forty-two processes taken through four maturity levels, every day attributed to the method that removed it, and which methods return the most.	Predict your own result. It is modelled, not measured, and it says so.

The order they go in

1	**Map it** — in PFV. Find the biggest wait.
2	**Diagnose it** — this page. Name the mechanism, or admit you cannot yet.
3	**Test it** — the bottleneck checks in the rubric. Is this a capacity constraint or something cheaper? Getting this wrong is how organisations hire into a queue that was never short of people.
4	**Choose the fix** — the rubric ranks the mechanisms that act on your kind of wait and says what each looks like at your maturity level.
5	**Size it, if the money matters** — How to Improve for what this class of change has typically returned; your own toolkit's calculator for what it costs you.
6	**Commit a prediction** — the model, before you touch anything. Days removed, with a band, dated.
7	**Do one thing**, then re-measure with the same map and score the prediction.

> Steps 3 and 6 are the two people skip, and they are the two that protect you. The bottleneck test stops you buying capacity you did not need; the committed prediction is what makes the next conversation about evidence rather than opinion.

Getting real value from How to Improve

It is a reference set, not a forecast, and it earns its keep in three specific situations.

Before a conversation about headcount. Across the forty-two worked processes, the single method that returned the most — self-service replacing request-and-wait — accounted for a fifth of every day saved, from twelve applications. None of the top methods was a purchase. If somebody is proposing to hire into a queue, that distribution is the argument.

When you are asked what improvement is worth. It gives you a defensible shape rather than a number you invented: a typical process lost about 71% of its lead time across three passes, the first pass usually did the most, and hands-on work barely moved throughout — all of the reduction was waiting removed.

When somebody claims AI will fix it. The same set run with the work automated removes 80% of the work and only 66% of the waiting, and on 22 of 42 processes fixing the flow with the same people came out ahead. That is not an argument against automation; it is the reason to map before deciding, because the map says which parts are even eligible.

> What it will not do is tell you what *your* process will return. The weights are fitted to modelled samples, the page says so, and the model page exists precisely so your own numbers can start replacing them.

Where this fits. The white paper argues the method; the tool guides say which control to press; this page is the reasoning between a map and a decision. The tools cost nothing to use and keep everything in your browser.