

BEYOND THE LESSON LOG

Programme Retrospectives Fail When Nobody Can Change the System

Teams turn evidence into experiments; programmes too often turn it into archives



Programme Retrospectives Fail When Nobody Can Change the System

Programme · No. 168

Originally written May 2012 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

“A retrospective without the authority to change its subject is a history meeting.”

The Lesson That Returned Every Quarter

At the end of a difficult release, a programme gathered thirty-two people for a retrospective. The walls filled with notes: test environments arrived late, business decisions took too long, supplier hand-offs were unclear, and data defects were discovered after system testing had begun.

The session was candid. The themes were sensible. A lessons report was circulated within a week.

Three months later, the next release encountered the same four problems in the same order.

This was not because people had failed to learn. The delivery teams had learned quite a lot. One team changed the order of its testing, paired an analyst with a developer on ambiguous stories, and reduced reopened defects from twenty-three to nine in a single iteration. The programme, however, converted every issue that crossed a team boundary into a recommendation for someone else.

That is why retrospectives work so readily inside teams and fail so predictably across programmes. A team retrospective connects observation to people who can alter the work. A programme retrospective often connects observation to a governance structure designed to record rather than change.

Learning Needs a Controllable System

The team retrospective has a practical advantage that is easy to underestimate: proximity.

The people in the room performed the work. They can usually see the sequence that produced the problem. They can choose a small change, try it quickly and inspect the result. If a daily build fails because code is integrated too late, the team can change its own practice tomorrow. The feedback loop is short and ownership is visible.

Programme problems are different in kind.

- The environment delay belongs to an infrastructure function serving ten projects.
- The slow decision belongs to a sponsor with several competing responsibilities.
- The supplier hand-off is shaped by a contract agreed before the team formed.
- The data defect originates in an operational process outside the programme.

No delivery team can correct these conditions alone. When a retrospective treats them as if they were merely team behaviours, it produces frustration. When it records them as “lessons for future programmes”, it produces archives.

The missing element is not insight. It is authority over the system that created the insight.

A retrospective without the authority to change its subject is a history meeting.

Why Programme Lessons Become Harmless

Programme leadership often believes that aggregation will create influence. Team lessons are collected, grouped into themes and presented to a board. Yet each step can weaken the evidence.

A specific observation—“the acceptance environment was unavailable for seventeen working days because no budget owner would approve additional storage”—becomes “environment planning should begin earlier”. The named delay disappears. The unmade decision disappears. The responsible authority disappears. What remains is universally agreeable and operationally useless.

This softening is not accidental. Precise lessons create discomfort because they reveal that many programme failures are consequences of organisational choices. A board can endorse “improve communication” without changing a mandate. It cannot endorse “give the release manager authority to allocate shared test capacity up to an agreed limit” without redistributing power.

The programme therefore becomes very good at learning language and very poor at learning behaviour.

A composite portfolio review illustrated the pattern. Twelve projects submitted 146 lessons over a year. Seventy-eight concerned environments, business availability or cross-project dependencies. The central office categorised them, published quarterly summaries and added three fields to the reporting template. Not one capacity decision, role definition or supplier obligation changed. The following year produced 139 lessons, with almost the same distribution.

The database was growing. The organisation was standing still.

The Serious Case for Caution

There is a strong argument for keeping programme retrospectives advisory. Large programmes contain competing interests, contractual limits and consequences that a delivery team cannot see. A local group may recommend dedicated infrastructure because shared capacity caused delay, while the organisation reasonably concludes that dedicated capacity would be too costly. A team may want immediate business decisions, while the sponsor must reconcile policy, finance and operational implications.

Not every frustration is evidence that authority should be delegated. Not every repeated complaint identifies the right remedy. Central leadership must compare patterns, test costs and protect the wider organisation.

That objection is valid. But caution does not justify a lesson process with no decision point. If a recommendation is rejected, the programme should know why. If the cost is too high, the exposure should be accepted explicitly. If the evidence is insufficient, the next release should gather it deliberately.

The failure is not that every retrospective demand goes unmet. The failure is that nobody is required to decide.

Turn Reflection into an Intervention

A programme retrospective should be designed around a small number of system changes, not a long inventory of observations.

The discipline begins by preserving the mechanism. Name the event, the sequence, the delay and the consequence. Avoid translating a hard fact into a general aspiration.

Then separate issues by sphere of control.

Sphere	Proper response
Inside one team	Team chooses an experiment and reviews the result
Across teams	Programme leader assigns an owner and common change
Outside the programme	Sponsor or board makes an explicit decision
Contractual or structural	Escalate with cost, consequence and options

For every cross-boundary lesson, the board should receive a decision-ready statement:

1. What recurred? Describe the specific pattern and frequency.
2. What did it cost? Show elapsed time, rework, missed scope or operational exposure.
3. What mechanism produced it? Identify the decision, dependency, incentive or capacity constraint.
4. What can change before the next cycle? Offer a bounded intervention, not an abstract recommendation.
5. Who must decide by when? Put authority and time on the record.

This is more demanding than a lessons workshop because it makes reflection consequential. It also means fewer lessons should survive. Ten weak observations do not equal one change that removes a recurring source of delay.

The Programme Must Be Willing to Learn About Itself

Retrospectives became popular because they gave teams permission to inspect their own working system rather than blame individuals. Programmes have adopted the meeting but often withheld that permission from the wider organisation.

They invite teams to speak, then exclude funding, governance, supplier terms and executive availability from the field of change. The most important causes are declared fixed before the conversation begins.

A programme that genuinely wants to learn must be prepared to discover that the defect sits in its own design. The cadence may be wrong. The board may meet too rarely. Shared services may have accepted more demand than they can serve. A contract may reward completion of documents while hindering collaboration. These are not comfortable retrospective topics, but they are precisely where programme-level improvement resides.

Team retrospectives succeed because the loop from fact to experiment is short. Programme retrospectives will succeed when the loop from fact to authority becomes equally clear.

Until then, programmes will continue to celebrate candour, publish lessons and repeat them on schedule.