



BEYOND PROJECT ECONOMICS

Platform Teams Cannot Succeed on Project Economics

The structural conflict between temporary delivery and enduring cloud capability

Platform Teams Cannot Succeed on Project Economics

Programme · No. 187

Originally written July 2016 | Re-edited and re-rendered for the WhereBeyond Edition, 2025

"A platform is a promise of continuity funded by an organisation addicted to temporary work."

1. The argument begins in the budget meeting

The platform team asks for twelve months of funding to build a shared cloud foundation: identity, network connectivity, machine images, monitoring, automated provisioning and support. The investment committee asks which projects will use it and how much each will contribute.

Three project sponsors agree that a common platform is strategically important. The first will fund only the network component because its application is already late. The second wants a different security pattern. The third will not commit until the platform proves it can meet a deadline that the platform cannot prove without a real workload.

By the end of the meeting, the organisation has not rejected the platform. It has divided it into project charges.

Six months later, three projects have built three versions of the same foundation. The central platform team is criticised for moving too slowly. The projects are criticised for creating duplication. Everyone is correct within the logic of their own structure.

This is the quiet conflict beneath many cloud programmes in 2016. We speak about reusable platforms, continuous delivery and common controls, but we continue to fund, govern and reward temporary projects. The platform team is asked to create continuity inside a management system designed for completion.

A platform is a promise of continuity funded by an organisation addicted to temporary work.

2. Two legitimate forms of urgency

The tension is often presented as a clash between enlightened platform thinking and short-sighted project delivery. That is too easy.

Project teams have a legitimate urgency. They carry a defined outcome, a budget, a date and a sponsor who expects progress. Their risks are visible and immediate. A delayed application may hold up a product launch, a regulatory response or a

data-centre exit. When a platform team says, “Wait for the standard service,” the project hears, “Accept our schedule in place of yours.”

Platform teams have a different, equally legitimate urgency. They can see that each local exception increases the cost of every future change. A second identity pattern means another access model to assure. A bespoke network route creates another dependency to monitor. A manually configured environment becomes another configuration nobody can reproduce with confidence.

The project experiences delay now. The platform experiences entropy later.

Both are rational. The failure lies in an organisation that asks them to resolve a structural trade-off through goodwill.

3. The project is temporary; the dependency is not

The project model is powerful because it creates concentration. People, funding and authority gather around a specific change. Ambiguity is converted into scope, milestones and accountable delivery. For work with a clear end, this is an effective instrument.

Cloud platforms do not have a clear end.

A platform serves multiple applications at different stages. It must be secure before the first workload, economical after the tenth, operable during an incident and adaptable when requirements change. Its value grows through reuse, but its costs arrive before reuse is proven. It requires maintenance after the programme that created it has closed.

The mismatch produces several predictable distortions.

- Projects optimise the entry point. They need the capability required for their release, not the broader service needed by later users.
- Platforms optimise the common case. They resist local variation because every exception becomes enduring support work.
- Funding follows change, not stewardship. Capital is available for migration, while routine maintenance, service improvement and technical debt compete for operational budgets.
- Governance asks for certainty too early. Sponsors want a complete service catalogue and fixed cost before usage patterns are known.
- Benefits are counted locally. The project records its delivery outcome; avoided duplication and faster future provisioning sit between business cases.

The result is not merely inefficient architecture. It is confused accountability. The platform team owns common components but may not control the projects that consume them. The project owns delivery but may not own the operational consequence of its exception.

4. The platform becomes either a gate or a help desk

When the structural tension is not resolved, platform teams tend to fall into one of two identities.

The first is the gatekeeper. It publishes standards, controls access and requires projects to conform. This protects coherence, but the team becomes a queue. Projects learn to escalate, obtain waivers or build around the platform. The platform's authority grows while its credibility falls.

The second is the internal help desk. It responds quickly, creates environments, fixes pipelines and absorbs every variation. Projects are pleased, but the platform becomes a collection of bespoke services held together by scarce specialists. Its responsiveness today becomes fragility tomorrow.

Neither identity is sustainable.

A platform must be opinionated enough to create leverage and responsive enough to earn adoption. It needs the right to say no to variation and the obligation to make the standard path genuinely useful. That balance cannot be created by a mission statement. It depends on funding, measures and decision rights.

5. The seductive idea of the internal product

One response is to describe the platform as an internal product and project teams as its customers. This is a useful correction. It encourages the platform team to understand user needs, publish a service catalogue, improve experience and measure adoption rather than mere compliance.

But the language can mislead.

A customer in an external market can often choose another supplier and bear the consequence. An internal project may be required to use the platform, while the platform team may depend on that project's budget. The relationship contains authority as well as service.

Nor is every request evidence of unmet customer need. A project may ask for a custom pattern because its design is unusual, because the standard genuinely fails, or because changing its plan is politically harder than asking the platform to change. Product language does not distinguish these cases automatically.

The internal-product idea is strongest when it changes behaviour:

- the platform publishes clear capabilities and service levels;
- consuming teams can see the roadmap and influence priorities;
- adoption friction is treated as a platform defect, not user resistance;
- exceptions reveal either a legitimate new pattern or a local choice that must carry its own cost;
- the platform team remains accountable for operability after the project closes.

It is weakest when “product” becomes a new label for the same central infrastructure function.

6. The strongest case for project freedom

There is a serious argument against strong platform control. Cloud technology is changing quickly. A central team can become conservative, protect its chosen patterns and force every application into the lowest common denominator. Delivery teams closer to users may innovate faster and understand their needs better. Some duplication is a reasonable price for learning, particularly while the organisation is still discovering what cloud can offer.

This argument deserves more than dismissal. Premature standardisation can freeze yesterday's assumptions. A platform built too far ahead of real demand can become expensive shelfware. Local experimentation is often how better patterns emerge.

The answer is not to eliminate project freedom. It is to distinguish exploration from unowned divergence.

An experiment has a question, a limited scope, an owner and a point at which evidence will be reviewed. Unowned divergence is simply a permanent exception created under deadline pressure. The first expands organisational knowledge. The second expands the support estate.

The platform should make experimentation safe and visible. Projects should not be forced to wait for central certainty, but they should be required to declare when they leave the standard path, who will support the result and when the pattern will be accepted, retired or incorporated.

7. A small example of a large problem

Consider a composite organisation with four application migrations planned over nine months. The platform team offers a standard environment built in ten working days. The first project needs a non-standard network inspection rule and estimates that waiting for central design will add three weeks. It builds its own environment in eight days.

The project meets its development milestone.

Operations later identifies 27 manually configured rules, two shared administrative accounts and monitoring that reports server availability but not transaction failure. Bringing the environment into the support model takes 19 additional working days. The apparent three-week saving has moved downstream and become less visible.

The second project sees the conflict and waits for the platform. Its start is delayed because the central team has only one network specialist. The sponsor escalates and receives priority, displacing work for the third project.

Nothing here is caused by bad intent. The platform is underfunded before demand is proven. Projects are measured against their own milestones. Operations enters after design. Escalation becomes the informal prioritisation mechanism.

The structural correction is not another workshop on collaboration. It is to fund the network pattern as a shared capability, put operational acceptance into the project's definition of completion, publish capacity and queue time, and give one forum authority to choose between project urgency and platform integrity.

Once the decision is visible, disagreement becomes governable.

8. Transformation intent meets annual planning

Organisations frequently announce a cloud-first strategy while retaining annual planning assumptions that favour projects.

A project business case can describe scope, cost and expected benefit. A platform business case depends on uncertain future consumption. Ask it for the same precision and it will either understate what is needed or invent confidence.

This is where transformation intent meets financial reality. Leaders say they want reuse, but ask the first project to pay the full cost. They want speed, but fund platform capacity only after a queue appears. They want standardisation, but allow each sponsor to escape it when a deadline becomes uncomfortable.

The platform then carries an impossible mandate: be ready before demand, but spend only after demand; enforce common patterns, but do not delay any project; provide enduring service, but rely on temporary funding.

We should recognise this contradiction as a design problem, not a performance problem.

9. A different compact between platform and project

The tension cannot be removed, because local urgency and shared integrity will always compete. It can be made productive through a clearer compact.

9.1 What the platform owes

- A small set of usable, documented standard paths.
- Transparent lead times, service levels and constraints.
- A roadmap shaped by real demand rather than technical enthusiasm.
- Evidence that common controls reduce work for consuming teams.
- An explicit exception path with timely decisions.
- Enduring ownership of the shared service.

9.2 What the project owes

- Early declaration of demand and non-standard requirements.
- Use of the standard path where it meets the need.
- Funding or capacity for genuinely project-specific additions.
- Operational evidence and support ownership for exceptions.
- Participation in testing and improving shared patterns.
- Acceptance that local deadlines do not erase enterprise consequence.

9.3 What leadership owes both

- Core funding for the minimum viable platform before projects arrive.
- A prioritisation mechanism that is not based on escalation volume.
- Measures spanning project delivery and enduring operability.
- Authority to retire duplicate patterns.
- A deliberate allocation for experimentation.
- A decision on trade-offs that teams cannot resolve themselves.

The platform-project relationship works when reuse is funded as an enterprise choice and exceptions are treated as conscious investments, not private escape routes.

10. The deeper question is what the organisation believes change is

Platform teams and project teams represent two different beliefs about transformation.

The project belief says change is an intervention: mobilise, deliver, transfer and close. The platform belief says change creates an enduring capability that must continue to evolve. Neither belief is universally correct. A facility move is not the same as an internal service used by dozens of applications. A one-off legal remediation is not the same as a common deployment path.

The trouble begins when the intervention model is applied to enduring capability.

Then the organisation celebrates the launch of a platform while nobody owns its second year. It counts migrated applications while common services accumulate exceptions. It closes the programme while the team that understood the design disperses.

The cloud debate is making this visible because infrastructure is no longer merely a fixed asset purchased in advance. It is becoming a set of services that can be consumed and changed continuously. Our organisational design has not caught up with that shift.

11. The structural story

The conflict between platform and project teams persists because it is sustained by the organisation's deepest routines:

- temporary funding;
- local business cases;
- functional accountability;
- milestone-based assurance;
- annual budgeting;
- operational ownership assigned late.

Asking teams to collaborate better without changing those routines is asking them to behave against the system that judges them.

The platform team should not defeat the project team, nor should projects be forced into central obedience. The real work is to create a structure in which short-term delivery and long-term coherence are both represented in the same decisions.

That means funding a minimum common foundation, giving platforms enduring ownership, allowing bounded experimentation, pricing the operational consequence of exceptions and measuring the service after the project milestone has passed.

It also means accepting that some tension is healthy. Platforms that never face impatient projects become self-referential. Projects that never face platform constraints externalise their costs. Each reveals the other's blind spot.

The mature organisation does not seek harmony. It builds a fair argument and gives that argument somewhere to be decided.

Cloud transformation will not be proven by the number of platforms launched or projects migrated. It will be proven when the organisation can build shared capability without making every project wait, and deliver urgent change without making every future team pay for it.