TECH ROI WEEKLY

From Projects to Platforms

The Operating Model That Makes Technology Investments Compound

A practical framework for persistent ownership, reuse and platform value

Chris Williams | CTO & Enterprise Architect

linkedin.com/in/chriswilliams1965

Publication Integrity Note

This article does not present invented client case studies, fabricated savings figures or generic ROI percentages. Where external research is referenced, the source is identified and the claim is limited to what the source supports. Examples of platform capabilities are category examples, not claims about a named organisation. Financial treatment of technology expenditure depends on accounting policy, jurisdiction and the nature of the work, so this article does not assume that platform funding must always be classified as operating expenditure.

OPENING

The project model of technology investment has a structural weakness: it is designed around completion.

A project has a start date, a budget, a scope and an end state. Those controls are useful when the work genuinely has a defined finish. They become problematic when the thing being funded is a long-lived digital capability that must continue to change after the first release.

The software does not stop ageing when the project closes. Customer expectations continue to move. Security requirements change. Dependencies are upgraded. Regulation evolves. Data volumes grow. New products need access to the same capability.

Yet many organisations still separate build from run, disperse the delivery team after launch, and make future improvements compete for a new round of project funding.

That creates a familiar cycle:

Fund. Build. Launch. Hand over. Accumulate friction. Fund again.

A product and platform operating model changes the unit of management. Instead of organising technology primarily around temporary initiatives, it organises persistent teams around products and reusable platforms, with continuing accountability for outcomes.

This is not simply an agile terminology change. It changes ownership, prioritisation, funding decisions, team boundaries and measures of success.

McKinsey's 2023 Operating Model Index research, based on a survey of senior leaders from more than 400 publicly traded companies, found that companies with more mature product and platform operating models were associated with stronger business outcomes. The most mature group had 60 percent greater total shareholder returns and 16 percent higher operating margins than the bottom half of the sample. McKinsey presents these findings as correlation, not proof that the operating model alone caused the financial results.

The strategic question is therefore not, "Should every project become a platform?"

It is, "Which technology capabilities need persistent ownership and reuse so that each new investment builds on what already exists?"

1. WHY THE PROJECT MODEL BREAKS DOWN FOR LONG-LIVED TECHNOLOGY

Project governance exists for good reasons. It gives leaders a mechanism for approving expenditure, controlling scope, coordinating dependencies and managing delivery risk.

The problem is not the existence of projects. The problem is applying project logic to capabilities that do not have a natural completion point.

1.1 Delivery can become the success measure

Project governance commonly tracks scope, schedule, budget and delivery milestones. Those are important controls, but they do not by themselves demonstrate business value.

A system can launch on time and still fail to achieve adoption. A data capability can meet its technical acceptance criteria and still create poor-quality outputs. A developer tool can be deployed across the estate and still add friction if teams avoid using it.

For long-lived digital products and platforms, delivery is a milestone. It is not the outcome.

Success requires evidence after launch: adoption, reliability, user experience, cost, time saved, revenue enabled, risk reduced or another measurable business result appropriate to the capability.

1.2 Handover fragments accountability

A traditional build-and-run separation can leave responsibility distributed across delivery teams, operations teams, vendors, architecture, security and business owners.

When nobody owns the full lifecycle, improvement becomes episodic. Technical debt competes with feature requests. Operational issues are treated separately from product decisions. Architecture decisions are revisited only when a major programme is approved.

Persistent product and platform teams reduce that fragmentation by keeping accountability with a team that owns the capability over time.

1.3 Reinvestment becomes exceptional

A long-lived technology capability needs ongoing investment. Some of that work is visible, such as new features. Some is less visible but equally important: dependency upgrades, automation, resilience, security remediation, observability, documentation and simplification.

If every improvement requires a new project, the organisation creates transaction cost around routine stewardship. Work is delayed until it becomes urgent enough to justify a new initiative.

A persistent team can instead manage a continuous backlog and make trade-offs between new value, reliability, technical health and cost.

The key distinction is simple: projects are useful for bounded change; products and platforms are useful for capabilities that must continue to evolve.

2. WHAT A PLATFORM IS, AND WHAT IT IS NOT

McKinsey describes platforms as back-end technology and data capabilities that support products and the enterprise more broadly. Thoughtworks similarly argues that internal platforms should be treated as products with defined customers, roadmaps and feedback loops.

A platform is not defined by its technology stack. It is defined by reuse and service to consumers.

A shared Kubernetes cluster is not automatically a platform. A data lake is not automatically a platform. An API gateway is not automatically a platform.

They become platform capabilities when they are intentionally designed, operated and improved so that other teams can consume them effectively.

2.1 Four characteristics of a useful platform

Persistent ownership

A named team owns the platform lifecycle, including roadmap, reliability, usability, cost and technical health.

Defined consumers

The platform has identifiable internal customers. The team understands their needs and measures adoption and friction.

A service interface

Consumers can access the capability through a repeatable interface such as APIs, self-service workflows, templates, documented services or managed components. Reuse should be easier than rebuilding.

Continuous evolution

The platform changes in response to consumer demand, technology change, security requirements and operational evidence. It is managed as an enduring capability rather than a one-off deliverable.

2.2 Typical enterprise platforms

Examples include:

Platform capability	What it can provide
Data platform	Governed ingestion, storage, transformation and access to data
Identity platform	Authentication, authorisation and single sign-on services
Developer platform	Build, test, deployment and self-service engineering workflows
Observability platform	Logging, metrics, tracing, alerting and operational insight
API platform	API discovery, management, security, versioning and policy enforcement
Security platform	Shared security controls, secrets, scanning and policy automation
Integration platform	Reusable messaging, event and integration capabilities

Not every shared component deserves its own platform team. The value comes from reducing duplicated effort and cognitive load for consuming teams, not from creating another organisational layer.

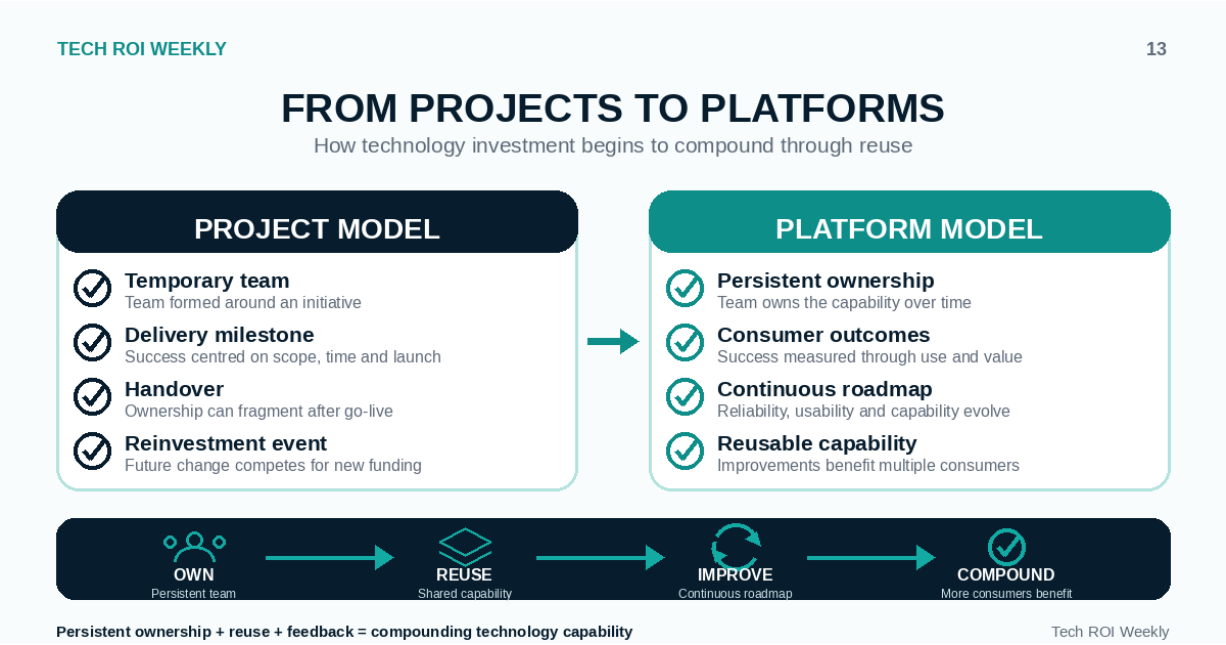

Figure: The operating-model shift from temporary project delivery to persistent platform ownership and reuse.

3. DESIGNING THE PLATFORM OPERATING MODEL

A platform model requires more than assigning a new label to existing teams. The operating model must make ownership and decision rights explicit.

3.1 Start with the value stream, not the technology catalogue

The first question should be: where are delivery teams repeatedly solving the same problem?

Look for capabilities that are:

- Needed by multiple product or stream-aligned teams
- Expensive or risky to reproduce independently
- Important to security, reliability, data quality or delivery speed

- Suitable for a repeatable service interface
- Valuable enough to justify persistent ownership

The aim is not maximum centralisation. It is deliberate reuse where reuse reduces friction.

3.2 Use Team Topologies as a design lens

Team Topologies, by Matthew Skelton and Manuel Pais, distinguishes stream-aligned teams from supporting topologies including platform and enabling teams. Its central idea is to optimise for fast flow of value while managing team cognitive load.

A platform grouping exists to provide capabilities that stream-aligned teams can consume without having to understand or operate every underlying detail. The platform should therefore be judged partly by how much complexity it removes from its consumers.

An enabling team has a different purpose. It helps another team acquire a capability or practice and should not become the permanent owner of that team's work.

This distinction matters because platform transformation often fails when every central specialist group is renamed as a platform team without changing how it serves consumers.

3.3 Treat the platform as a product

Thoughtworks has advocated product thinking for internal platforms for years. The practical implication is that the platform team needs product management disciplines:

- Identify target consumers and their priority journeys
- Maintain a roadmap based on consumer needs and strategic direction
- Observe how teams actually use the platform
- Make self-service paths easy to discover and adopt
- Measure friction as well as technical performance
- Retire capabilities that no longer justify their cost or complexity

The internal customer relationship is critical. A technically elegant platform that teams bypass is not delivering its intended value.

3.4 Keep team size evidence-led

There is no defensible universal rule that a platform serving a particular number of teams requires a specific headcount. Team size depends on the breadth of the platform, service criticality, level of automation, support model, engineering maturity and number of consumer journeys.

Use workload and outcome evidence instead:

- Is the team able to maintain reliability and security?
- Is roadmap work progressing at an acceptable rate?
- Are consumers waiting for support or platform changes?
- Is toil consuming engineering capacity?
- Are ownership boundaries clear enough to avoid coordination overload?

Scale the team or split the platform only when evidence shows the current design is constraining outcomes.

4. FUNDING: MOVE FROM INITIATIVE CONTROL TO CAPABILITY CONTROL

Funding is one of the hardest parts of the transition because annual project portfolios are embedded in finance, procurement and governance processes.

The objective is not to eliminate financial control. It is to align control with persistent accountability.

4.1 Fund teams against outcomes and guardrails

McKinsey's product and platform research emphasises tying funding to measurable goals and progress. Rather than approving a complete multi-year feature scope up front, leadership funds a product or platform team against defined outcomes, strategic priorities and financial guardrails.

The team then manages the backlog within those boundaries.

This preserves governance while reducing the need to create a separate funding event for every incremental change.

4.2 Choose an allocation model deliberately

Common approaches include:

Central funding: shared platforms are paid for from a central technology budget. This can simplify adoption for foundational services such as identity or security.

Showback: consuming teams can see the cost of the capacity or services they use, even when no internal charge is posted. This improves transparency and helps teams understand demand.

Chargeback: costs are allocated to consuming teams through an agreed internal charging method. This can strengthen accountability but can also create friction if the charging basis is unstable or difficult to understand.

There is no universal maturity sequence that every organisation should follow. The right model depends on financial governance, platform type, organisational incentives and the reliability of consumption data.

4.3 Do not confuse funding model with accounting classification

A persistent platform can be funded continuously without assuming that all associated expenditure is operating expense. Accounting treatment depends on the work performed, applicable standards and organisational policy.

Technology leaders should work with finance to separate the operating-model question, "How do we sustain the capability?", from the accounting question, "How should this expenditure be classified?"

5. MEASURING WHETHER THE PLATFORM IS WORKING

Platform metrics should connect technical quality to consumer outcomes.

A useful scorecard can include:

Measure	What it tells you
Adoption	Whether eligible teams choose and continue to use the platform
Time to first use	How quickly a new consumer can obtain useful capability
Consumer satisfaction	Whether the developer or internal user experience is improving
Reliability	Whether the platform is dependable enough for its consumers
Support demand and toil	Where friction or manual work is accumulating
Delivery enablement	Whether consumers can deliver changes faster or with fewer repeated tasks
Unit cost	Whether the cost of delivering a defined unit of platform service is improving
Reuse	Whether shared capabilities are replacing duplicated local solutions

Avoid treating one metric as the answer. High adoption can be misleading if use is mandatory and satisfaction is poor. High reliability can be misleading if onboarding takes weeks. Low unit cost can be misleading if the platform is failing to meet required controls.

The scorecard should make trade-offs visible.

6. A PRACTICAL TRANSITION PATH

A platform operating model should be introduced through evidence, not a wholesale reorganisation based on theory.

Step 1: Find one repeated source of friction

Select a capability where multiple teams are independently solving the same problem or waiting on the same bottleneck. Good candidates can include deployment, observability, identity, data access or integration.

Document the current experience before changing it. Measure waiting time, repeated work, support demand, failure modes and cost where possible.

Step 2: Establish persistent ownership

Create a clear owner and team boundary. Define who the consumers are, what service the platform will provide, what it will not provide and how consumers will request changes.

Do not begin with a large platform build. Begin with the smallest reusable capability that removes a real constraint.

Step 3: Build a paved path

Provide a supported route that is easier than every team building its own solution. The paved path should include automation, documentation, policy guardrails and self-service where practical.

Adoption should be earned through usefulness wherever possible. Mandates may be necessary for security or regulatory controls, but a mandated platform still needs good usability.

Step 4: Measure consumer outcomes

Track whether teams onboard faster, repeat less work, encounter fewer avoidable failures or gain access to capabilities they previously could not justify individually.

Use those results to decide whether to expand the platform, improve it or stop investing.

Step 5: Scale the model selectively

Once one platform demonstrates value, identify the next shared capability using the same criteria. Do not turn the organisation into dozens of platform teams simply because the first one worked.

The target state is a portfolio of platforms that remove repeated complexity and support strategic products, not a platform organisation for its own sake.

7. BUILDING THE BUSINESS CASE WITHOUT INVENTED ROI

A platform business case should be based on measured reuse and avoided duplication, not a generic percentage improvement.

Start with the current baseline:

- How many teams are building or operating similar capability today?
- How much engineering time is consumed by repeated setup, maintenance or support?
- What tooling, infrastructure or vendor cost is duplicated?
- What delivery delays arise from waiting for specialist teams or manual approvals?
- What incidents or control failures are linked to inconsistent local implementations?

Then define the change that the platform is expected to make:

- Which repeated activities will become self-service?
- Which components will be standardised and reused?

- Which local solutions can be retired or avoided?
- Which risks will be reduced through shared controls?
- Which new capabilities become economically practical because they are shared?

Measure realised value after adoption. A defensible calculation might include verified engineering time released, duplicated services retired, support workload reduced, onboarding time improved and risk controls automated.

Do not count theoretical reuse. Count actual consumption.

Do not claim the same benefit twice through both time saved and headcount reduction unless both actually occurred.

Do not treat platform adoption as value in itself. Adoption matters because of the outcome it enables.

This is how technology investment begins to compound: each new product can consume capabilities that have already been funded, tested and improved, while later platform investment benefits a larger base of consumers.

8. COMMON FAILURE MODES

Renaming projects as platforms

If funding still ends at delivery, ownership still fragments at handover and the team has no consumer feedback loop, the operating model has not changed.

Building before discovering demand

A central team can spend months building a technically sophisticated platform that solves the wrong problems. Start with consumer friction and validate demand before expanding capability.

Forcing adoption without improving experience

A mandate can increase usage statistics while creating hidden workarounds and resentment. Standardisation should reduce effort for consumers, not merely transfer control to a central team.

Treating the platform as a ticket queue

If every consumer request requires manual intervention from the platform team, the platform becomes another dependency bottleneck. Use APIs, automation, templates and self-service to convert repeated requests into reusable capability.

Measuring only uptime

Reliability is essential, but it is not enough. A platform can be highly available and still be difficult to discover, expensive to consume or slow to change.

Assuming every capability should be centralised

Some capabilities are better owned directly by stream-aligned teams. Centralisation creates its own coordination cost. A platform is justified when the benefits of reuse and reduced cognitive load outweigh that cost.

CLOSING

The choice is not projects versus platforms in absolute terms.

Projects remain appropriate for bounded initiatives with a real finish. Platforms are appropriate where a capability must be reused, operated and improved for as long as the organisation depends on it.

The economic advantage comes from persistent ownership and reuse.

When a shared capability improves, every consuming team can benefit. When a new product launches, it can build on identity, data, deployment, observability, security and integration capabilities that already

exist. When a control changes, it can be implemented once in the platform rather than repeatedly across dozens of local solutions.

That is the compounding effect.

It is not guaranteed. A platform that nobody wants, cannot afford or cannot use easily simply creates a new layer of cost.

The leadership challenge is therefore to identify the capabilities where reuse has real economic value, fund them with persistent accountability, measure them through consumer outcomes, and stop investing when the evidence no longer supports them.

The question for every technology leader is:

Which capabilities should we build once, improve continuously and make reusable across the organisation?

KEY TAKEAWAYS

- Projects remain useful for bounded change, but long-lived capabilities need persistent lifecycle ownership
- Product and platform operating models organise teams around enduring outcomes rather than temporary initiatives
- McKinsey research found a strong association between operating-model maturity and business performance, but the research is correlational and should not be treated as a guaranteed ROI claim
- Team Topologies provides a useful model for reducing cognitive load on stream-aligned teams through appropriate platform and enabling team structures
- Internal platforms should be treated as products with defined consumers, roadmaps, feedback loops and measurable service outcomes
- Platform team size should be driven by workload, service criticality and consumer evidence, not a universal headcount formula
- Funding can be persistent while still retaining financial guardrails; accounting classification should be agreed separately with finance
- Measure adoption, time to first use, satisfaction, reliability, toil, reuse and unit cost together
- Build the business case from measured reuse, avoided duplication and realised outcomes, not generic savings percentages

RESEARCH AND REFERENCES

1. McKinsey & Company, "The bottom-line benefit of the product operating model", 19 December 2023. Research based on more than 400 publicly traded companies and the Operating Model Index.
https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/the-bottom-line-benefit-of-the-product-operating-model

2. McKinsey & Company, "Products and platforms: Is your technology operating model ready?", 28 February 2020.
https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/products-and-platforms-is-your-technology-operating-model-ready

3. McKinsey & Company, "The big product and platform shift: Five actions to get the transformation right", 9 June 2023.
https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/the-big-product-and-platform-shift-five-actions-to-get-the-transformation-right

4. Team Topologies, Matthew Skelton and Manuel Pais, IT Revolution, first edition 2019; official key concepts and team-type guidance.
https://teamtopologies.com/key-concepts-content/what-are-the-core-team-types-in-team-topologies

5. Team Topologies, "How Platform Teams Reduce Cognitive Load", June 2024.
https://teamtopologies.com/news-blogs-newsletters/2024/6/10/newsletter-june-2024-how-platform-teams-reduce-cognitive-load

6. Thoughtworks Technology Radar, "Platform engineering product teams".
https://www.thoughtworks.com/en-us/radar/techniques/platform-engineering-product-teams

7. Thoughtworks Technology Radar, "Applying product management to internal platforms".
https://www.thoughtworks.com/radar/techniques/applying-product-management-to-internal-platforms

8. Thoughtworks Technology Radar, "Incremental developer platform".
https://www.thoughtworks.com/radar/techniques/incremental-developer-platform

Author: Chris Williams, CTO and Enterprise Architect, Tech ROI Weekly

LinkedIn: https://www.linkedin.com/in/chriswilliams1965/