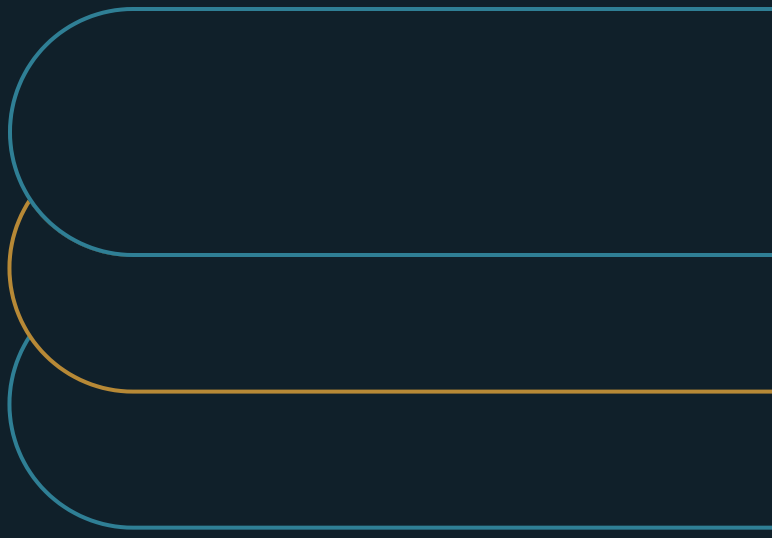
13

WHITE PAPER · PORTFOLIO & DELIVERY GOVERNANCE

Standardize the Component, Not the Project

How an EPMO turns repeated project learning into shared operating assets without erasing legitimate local variation



Marco Policani

Enterprise Portfolio · PMO · AI Operating Governance

policani.net · linkedin.com/in/marcpolicani

August 2026

EXECUTIVE SUMMARY

Promote a project component only when reuse, ownership, variants, change, exceptions, and retirement are visible.

Project learning rarely fails because organizations cannot store files. It fails because nobody decides which part of a successful project should travel, what may vary, who owns the shared part, or when it should change. This paper defines a project-to-standard review that promotes reusable operating components while protecting legitimate local judgment.

01 Standardize a bounded operating component — such as an intake question, decision record, readiness check, or handoff rule — rather than the whole project.

02 Promote only when the review can show repeated use, an accountable owner, approved variants, a version path, an exception signal, and a retirement trigger.

03 Treat local variation as design evidence: publish valid variants, bound exceptions, and route recurring departures into the next core decision.

04 Keep the EPMO in its lane: govern promotion and cross-project signals while subject owners retain authority over content and projects retain legitimate local choice.

CONTENTS

A project folder is evidence, not a standard

In one finance-critical release portfolio I supported, the useful breakthrough was not a new project method. It was a small set of readiness assets that could travel: entry questions, approved workback logic, test-case expectations, and a clear handoff into validation. Each delivery team still had its own scope, calendar, risks, and subject-matter experts. Shared entry questions stopped teams from

reopening what readiness meant at each handoff. The local parts protected the work from a false uniformity.

That distinction is easy to lose. A project succeeds, its documents are placed in a shared folder, and the organization calls the folder a standard. Another project copies the files, changes half of them, and creates a second version. Soon the repository contains several artifacts with similar names, uncertain owners, conflicting rules, and no clear way to know which one should be trusted. Activity has been preserved. Learning has not.

PMI's lessons-learned practice makes the gap visible. Learning should be captured throughout a project, then documented, analyzed, stored, retrieved, and converted into management action. At higher levels of practice, consistent input makes recurring issues easier to identify. Approved actions are tracked to completion.[1] A repository handles storage and retrieval. Analysis, action, and follow-through depend on management practice.

A government review provides an unusually direct warning. NASA had an agency-wide Lessons Learned Information System and required managers to review it. GAO still found no assurance that lessons were being applied to later missions, and found that lessons were not routinely identified, collected, or shared. GAO's remedies addressed management commitment, incentives, coordination, and mechanisms that made knowledge easier to use.[7]

The EPMO's opportunity is therefore smaller and more useful than standardizing projects. It is to notice which parts of project work keep recurring, decide whether those parts have earned reuse, and govern them as operating assets. The unit of learning is not the whole project plan. It is the component that can travel without carrying the entire project's context with it.

The reusable unit is smaller than the project

Projects combine common work with conditions that may never repeat. Sponsors, customers, and contracts differ. So do technologies, geographies, regulatory duties, risk levels, and delivery histories. Copying the project treats all of that context as if it were part of the reusable method. It produces a large template that local teams must either obey or quietly dismantle.

A component is narrower. It may be an intake question that exposes whether a sponsor can name the outcome. It may be a decision record that preserves the owner, evidence, alternatives, and next review point. It may be a source standard, a readiness check, an exception route, a measure definition, or a handoff rule. Each has a specific job. Each can be tested apart from the project that first produced it.

The boundary matters because reuse needs a stable core and an explicit edge. The core states what must remain true when the component moves. The edge states where local judgment begins. A decision record might always require a named decider and the evidence considered, while allowing

each portfolio to add regulatory, commercial, or technical fields. A readiness check might always establish entry conditions, while letting teams vary the tests that prove those conditions.

APQC's guidance on appropriate process standardization supports this design. It calls for standards that account for context-driven variants, with reference models and process governance helping to sustain the shared pattern.[3] The reusable core and its context-driven variants form one governed system.

This changes the EPMO's first question. Instead of asking, 'Which project template should everyone use?' it asks, 'Which operating component has repeated often enough that teams should not have to rediscover it?' The first question begins with control. The second begins with evidence.

Promotion must be earned

A useful artifact should not become a standard because an executive likes it, because a team delivered successfully once, or because the PMO needs a new template. Those signals show attention, not transferability. Promotion is a portfolio decision about whether the organization should rely on a component beyond the setting that created it.

EXHIBIT 1

Promotion turns local learning into a managed component

01

Iterate

Keep the candidate close to the work while its purpose, boundary, and failure modes emerge.

02

Mature

Test the core in another setting and record what stayed stable, varied, or failed.

03

Canonize

Publish the supported core, owner, scope, variants, version, and known limits.

04

Lock

Route changes through an explicit acceptance, variant, rejection, or evidence decision.

05

Monitor

Watch reuse, exceptions, workarounds, obsolescence, and signals that justify retirement.

The decision can be managed through five states: iterate, mature, canonize, lock, and monitor. These are not industry maturity levels. They are Marco's operating model for deciding when local learning becomes a shared asset.

Iterate: keep the component close to the work

A new component begins as a local answer. The team can change it quickly because its purpose, boundary, and failure modes are still being discovered. The evidence at this stage is practical: Did it remove a repeated ambiguity? Did people understand it? Did it expose a decision sooner? Did it create extra work somewhere else? A useful first result is not enough to promote it.

Mature: test the core in another setting

A component matures when another project can use its core without copying the original project's assumptions. The second use is especially valuable because it separates the essential fields from local furniture. Differences are recorded, not treated as defects. The review asks whether the change is a legitimate variant, a sign that the core is wrong, or evidence that the candidate is too narrow to share.

Canonize: publish the supported core and its boundary

Canonization is the explicit promotion decision. The component receives a name, purpose, accountable owner, current version, and boundary of use. Approved variants and known limits sit beside the supported core. A canonized intake question does not become a complete intake process. A shared decision record does not replace the judgment that fills it.

Lock: place change under governance

Locking places the shared core under visible change control. Proposed changes follow a defined path. The owner can accept a change into the core, approve it as a variant, reject it with a reason, or return it for more evidence. Version history protects users from finding out too late that the same asset meant different things in different projects.

Monitor: watch use, exceptions, and obsolescence

A component remains a standard only while it remains useful. Monitoring looks for repeated exceptions, workarounds, unclear handoffs, abandoned fields, stale sources, and changes in the setting that produced it. The outcome is not always another version. It may be a narrower scope, a split into two components, a return to local use, or retirement.

The promotion sequence protects both sides of the argument. It prevents a project office from imposing an untested answer, and it prevents proven learning from remaining trapped inside the team that discovered it. Most candidates should spend time in iterate or mature. Some should never move further. A small number should earn enterprise reliance.

The project-to-standard review

The review should be lightweight because its purpose is a decision, not a new ceremony. It can sit inside a regular portfolio, community-of-practice, quality, or operating-model cadence. The candidate owner brings a short evidence record. The review body decides one of four outcomes: keep local, return for more evidence, promote with a defined boundary, or retire.

Seven questions keep that decision honest.

- Repeated component. What exact part of the work has appeared in more than one project? Name the component, not the program that produced it.
- Evidence of reuse. Where has another team used the core, and what became easier, clearer, safer, or more consistent? Record failures and abandoned uses as well as successful ones.
- Owner. Who can explain the asset, decide a change, communicate a new version, and retire it? A repository administrator is not automatically the operating owner.
- Approved variants. Which differences are legitimate because context changes, and which would break the component's purpose or evidence standard?
- Version and change path. How will users know the current version, submit a change, see the disposition, and understand whether they must migrate?
- Exception signal. Which repeated workaround, override, failure, or local edit should force the owner to review the core or its boundary?
- Retirement trigger. What would show that the component has become obsolete, redundant, too costly to maintain, or unsafe to rely on?

The questions are deliberately harder than ‘Was the artifact uploaded?’ They force the candidate to show a pattern, a use boundary, and a management path. They also make absence visible. A component with no owner is a document. A component with no evidence of reuse is a proposal. A component with no exception signal is likely to drift. A component with no retirement trigger is a permanent obligation created by accident.

The review record should be short enough to read at the table. It should show why the candidate was promoted, who owns it, which version is current, and what will reopen the decision. The approved scope and variants sit beside that decision. Longer narratives remain supporting evidence.

Circulation is not evidence of reuse

A common promotion error is to count distribution. A file was downloaded, linked from an intranet, attached to several project plans, or mentioned in training. Those events show exposure. They do not show that the component shaped the work. A team can attach a standard decision record and still

make the decision in an unrecorded side meeting. It can copy a readiness checklist after the release is already committed. It can cite an intake form while bypassing the questions that matter.

Evidence of reuse should connect the component to an observable part of the operating process. Did another team use the intake questions before accepting demand? Did the decision record reach the accountable owner while alternatives were still open? Did the source standard change what was accepted as evidence? Did the handoff rule identify the receiver and entry condition before work crossed the boundary? These observations do not have to be large metrics. They have to show use at the point where the component was designed to matter.

The evidence should also preserve friction. If a team used the core but needed a local field, that is variant evidence. If the team abandoned the component because it arrived after the decision, that is timing evidence. If users completed it but could not explain its purpose, that is a design failure. Promotion based only on successful adoption stories will canonize components whose hidden cost is pushed onto local teams.

A useful review separates exposure from operating evidence. Encountering the component establishes adoption; applying it at the intended point establishes use. The stronger evidence shows a change in a named ambiguity, handoff, or control condition. Continued maintenance and access then show whether reuse can last. The review needs enough of this chain to avoid confusing page views with operating value.

This keeps the EPMO away from a false numerical threshold. Ten downloads are not weaker than one hundred if ten projects used the asset in live decisions. Two observed uses may be informative when the settings are materially different. Five nearly identical uses inside one team may teach very little about transfer. The review weighs relevance, variation, and consequence. It does not promote on volume alone.

A bounded operating illustration

The finance-critical readiness work mentioned at the opening shows what this looks like without turning one engagement into a universal case. The recurring problem was late entry into validation. Teams arrived with different release scopes, system changes, finance calendars, and test needs. A single end-to-end project template would have buried those differences or forced the teams to rewrite most of it.

The reusable component was the entry architecture. It asked whether the workback schedule was approved and the validation criteria were explicit. It also checked that test cases existed before formal testing. Engineering and finance had to resolve the questions that would otherwise surface in executive review. Those conditions could travel. The detailed test cases, dates, owners, and evidence remained local.

Reuse changed the component. Early versions exposed where the language was too broad and where a finance calendar created a legitimate variant. A shared knowledge base made the current material easier to find. Operating value appeared when teams used the readiness conditions before the handoff.

If this pattern were brought to a project-to-standard review, the promotion record would not claim that one checklist caused faster releases. It would show the repeated entry problem and the parts used across delivery teams. The same record would identify necessary local differences, the owner of the readiness definition, and the signals that would force a change. A change in finance policy, a repeated late exception, or teams bypassing a condition would matter more than the number of times the file was opened.

The illustration also shows why ownership must sit near the operating decision. A central PMO can steward the promotion record and see patterns across teams. It cannot decide the content of a finance validation rule by itself. The owner needs enough subject authority to judge the core, enough reach to communicate a change, and enough accountability to retire a rule that no longer protects the handoff.

Variation is design data

Local variation is often treated as noncompliance. That is a poor default. A project may vary because its sponsor has different authority, its data crosses a jurisdiction, its customer cannot accept the standard handoff, its technology has another failure mode, or its decision is harder to reverse. Those differences can be legitimate. They can also reveal that the shared core was defined too broadly.

PMI's research on governance of innovation is a useful counterweight to rigid standardization. Its case research across six large organizations describes a tension between formality and flexibility, and recommends governance tailored to each setting. It also argues for governance that is codified, understood, and consistently applied.[2] Consistency and flexibility are not opposites when the standard states where adaptation is allowed and who decides.

An approved variant is therefore part of the asset, not a tolerated deviation hidden in a project plan. It names the condition that justifies the difference, the part of the core that still applies, the owner who approved it, and the evidence that would end or broaden the variant. This turns local practice into information that the enterprise can learn from.

Exceptions play a similar role. One exception may be noise. The same exception across several projects is a signal. It may show that the core ignores a common dependency, that a field asks for evidence no team can obtain, or that users have found a safer route. The EPMO should not count exceptions only as breaches. It should aggregate them as design feedback.

This is also why a waiver log cannot sit apart from version governance. If exceptions accumulate but never reach the component owner, the enterprise maintains the appearance of a standard while every

project works around it. The published rule and the operating reality separate. Review of exception patterns closes that gap.

Give each departure a governed status

Local differences need more than a free-text explanation. The review should distinguish an approved variant from an exception and from evidence that the core must change. An approved variant repeats under a named condition and preserves the component's purpose. An exception permits a bounded departure for one case, with an owner and an expiry or return condition. A proposed core change shows that the shared rule no longer handles the work it was meant to govern.

Those statuses create different management actions. Variants are published so the next team can select them deliberately. Exceptions are watched for recurrence; a cluster may reveal a missing variant or a failing core. Proposed changes enter the version path and require evidence from the settings they would affect. Treating all three as local customization would hide the signal that makes the component improve.

A shared asset needs a lifecycle

Promotion creates an obligation. Once teams are told to rely on a component, someone must keep it current, explain its limits, and manage the consequences of change. The asset now has users, dependencies, and switching costs. Treating promotion as the finish line hides that burden.

MIT CISR's enterprise AI maturity work describes a move from pilots and capabilities to scaled AI ways of working. It also warns that there is no proven playbook for making that move.[4] The stage transition changes the unit of management: an isolated capability becomes a way of working with strategy, systems, roles, and stewardship around it.

The SEI maturity model makes that institutionalization more explicit. It emphasizes repeatable outcomes, workflow re-engineering, measurement, governance, operations, and continuous improvement. As a maturity model, it is not a controlled test of component reuse.[5] It supports the need for a managed capability without validating this paper's five promotion states.

When a shared component enters AI-enabled work, the lifecycle duty becomes sharper. A source standard, human review step, evaluation routine, logging rule, or escalation path may travel across several workflows. NIST's deployed-AI monitoring work warns that performance can degrade, logging can fragment, and human monitoring can become hard to scale. It also leaves cadence and use-case tailoring as open questions.[6] The right monitoring signal must follow the component's actual risk and context, not a universal calendar.

The same lifecycle logic applies outside AI, even when the signals differ. A financial readiness asset may change when close policy changes. An intake component may need revision when decision rights

move. A handoff rule may fail after a vendor, customer, or platform change. The owner must be able to see those shifts and act before a trusted component becomes a stale control.

Keep the EPMO in its lane

The EPMO should govern promotion, not author every component or approve every local choice. Subject-matter owners remain responsible for the substance. Project leaders remain responsible for fit in their setting. Risk, legal, security, finance, engineering, and operations keep their own authorities. An EPMO is positioned to provide the cross-project view and the decision system that connects those groups.

EXHIBIT 2

The EPMO connects the system without owning every decision

Project teams

Surface repeated components, evidence of use, local variants, and workarounds.

Promotion review

Decide whether to mature, canonize, revise, keep local, or retire the candidate.

Component owner

Own the supported core, approved variants, version path, and retirement decision.

Adopting teams

Use the current component, select valid variants, and report exceptions or failure.

The role begins with scanning projects for recurring questions, artifacts, workarounds, and failures. A promotion review then brings together the people who own the work and those who would reuse it. The EPMO keeps the core, approved variants, versions, and decisions visible, and watches for exception patterns that a single project may not see.

The EPMO should also resist two forms of empire building. It should not convert every useful local practice into an enterprise asset. Maintenance is real work, and low-frequency reuse may not justify it. It should not use the review to pull execution decisions upward. The purpose is to reduce repeated invention while preserving the project's authority over legitimate context.

Cadence follows the flow of candidates and the consequence of reliance. A large transformation portfolio may need a monthly review. A smaller environment may use quarterly or event-triggered decisions. High-risk components may need scheduled checks, while a low-risk checklist may change only when exceptions accumulate. The sources in this paper do not establish one correct timetable.

What this evidence cannot establish

The research base supports continuous lessons practice, context-aware governance, reference models with variants, institutionalized ways of working, and lifecycle monitoring. It also documents the failure of a large repository to assure application. It does not show that this promotion model produces a fixed return, that two reuses are enough, or that an EPMO should own every standard.

EXHIBIT 3

The evidence supports a practice boundary, not a fixed return

Lessons practice
PMI supports capture, analysis, retrieval, management action, and tracked follow-through.

Context boundary
PMI and APQC support codified practice with context-appropriate flexibility and variants.

Lifecycle evidence
MIT CISR, SEI, and NIST support managed capabilities, workflows, and monitoring.

Contrary evidence
GAO shows that an agency-wide lessons repository did not assure later application.

Not established
No source proves a promotion threshold, fixed return, universal cadence, or EPMO ownership.

Those gaps matter because a neat framework can create false confidence. Reuse evidence can be biased toward successful projects. Teams may comply with a component without using it in the decision. Owners may protect an asset they created. Exception counts may rise because reporting improved, not because the standard worsened. The review should treat those as interpretation problems, not hide them behind a score.

The framework is a governance test that makes the claims around a candidate visible and gives leaders a small set of valid dispositions. Thin evidence should keep the component local or send it back to mature. Restraint protects the standard from promotion by enthusiasm.

The standard is the managed component

Projects should not all mature the same way. Their evidence changes, dependencies move, and local conditions deserve judgment. Forcing them into one template confuses consistency with control and turns variation into paperwork.

The better enterprise move is to standardize the component that has earned reuse. Give it a stable core, a clear boundary, approved variants, an owner, a version path, an exception signal, and a retirement trigger. Then keep watching whether it still helps the work it was meant to serve.

A folder preserves what a project produced. A promotion decision establishes what the organization is prepared to rely on. The EPMO creates value in the distance between those two things.

Notes and sources

[1] Sandra F. Rowe and Sharon Sikes, Lessons Learned: Taking It to the Next Level, paper presented at PMI Global Congress 2006—North America, Project Management Institute, 2006. https://www.pmi.org/learning/library/lessons-learned-next-level-Communicating-7991

[2] Michael Knapp, Catherine P. Killen, Chris Stevens, and Shankar Sankaran, Governance of Innovation in Portfolios, Programs, and Projects, PMI Sponsored Research, January 31, 2019. https://www.pmi.org/learning/library/governance-innovation-projects-programs-portfolios-11796

[3] Mathias Kirchmer, Appropriate Process Standardization, APQC, January 23, 2024. https://www.apqc.org/resource-library/resource-listing/appropriate-process-standardization

[4] Stephanie L. Woerner, Ina M. Sebastian, Peter Weill, and Evgeny Káganer, Grow Enterprise AI Maturity for Bottom-Line Impact, MIT Center for Information Systems Research, August 21, 2025. https://cisr.mit.edu/publication/2025_0801_EnterpriseAIMaturityUpdate_WoernerSebastianWeillKaganer

[5] Ipek Ozkaya, Anita Carleton, Sebastián Echeverría, Robert Edman, John Haller, Erin Harper, Michael D. Konrad, Carol J. Smith, and Shawn Wray, AI Adoption Maturity Model v1.0, Carnegie Mellon University Software Engineering Institute, June 30, 2026. https://www.sei.cmu.edu/projects/ai-adoption-maturity-model-improving-how-organizations-adopt-ai-solutions/

[6] National Institute of Standards and Technology, NIST AI 800-4: Challenges to the Monitoring of Deployed AI Systems, March 9, 2026, updated March 18, 2026. https://www.nist.gov/news-events/news/2026/03/new-report-challenges-monitoring-deployed-ai-systems

[7] U.S. Government Accountability Office, NASA: Better Mechanisms Needed for Sharing Lessons Learned, GAO-02-195, January 30, 2002. https://www.gao.gov/products/gao-02-195

About the author

Marco Policani is an enterprise portfolio, PMO, and AI operating-governance leader. He builds portfolio governance systems where AI carries the analytical load and named humans own every decision — an approach documented across case studies, walkthroughs, and working governance guides on his portfolio site.

Portfolio & governance library: policani.net/governance · Full portfolio: policani.net · LinkedIn: linkedin.com/in/marcpolicani