FRAMEWORK #77

Dependency Residue

DEFINITION

Dependency residue is the economic, operational, contractual, or governance dependency that persists after a transformation program declares its originating system, platform, service, or capability eliminated, and that the program's cost model no longer counts.

FRAMEWORK FLOW

01
The Condition
A transformation program defines what is being eliminated — a platform, service, contract, or capability — and its cost model draws the boundary there.

02
The Boundary
The cost model stops counting the eliminated dependency. The boundary follows accounting lines: budget owner, invoice, contract.

03
Failure State
Runtime, contractual, operational, or governance paths still depend on it. The dependency follows execution and obligation lines, not the ledger.

04
Consequence
The residual dependency keeps generating cost or exposure after the program reports complete, outside any line item the program still owns.

Failure State: Counted Elimination

`Program declares it eliminated` → `Cost model stops counting it` →
`Calls, credentials, obligations, and data still depend on it` → `Program reports complete` →
`Residue bills outside any line item the program owns`

THE DIAGNOSTIC SIGNAL

LEDGER VS. RUNTIME

The ledger shows zero for the eliminated item. A runtime trace still shows calls to the old endpoint, an active credential, or egress from the old location.

UNCOUNTED SURVIVORS

Whatever survives the elimination — retention, contractual obligations, data that can't move on schedule — has no named owner and no budget line at closure.

TRIGGER CONDITION

Any transformation program that declares something gone and measures that claim against a cost model. Platform and direction of travel are irrelevant.

REMEDIATION SHAPE

Test the closure claim against the dependency boundary before closure. The target is zero uncounted residue, not zero residue: every survivor gets an owner and a line item.

ARCHITECTURAL RELATIONSHIPS

#143
Dependency Visibility Boundary
RELATED · MODERATE — CROSS-PILLAR

#143 asks whether a dependency surface can be seen at all; #77 asks whether a dependency that was seen is still counted once elimination is declared. Dependencies beyond #143's boundary are a common source of residue, but residue also forms around fully mapped dependencies the cost model excluded.

#137
Operating Model Transfer Gap
RELATED · MODERATE — CROSS-PILLAR

#137 measures the governance and authority that must be recreated on the target; #77 measures what the source still costs while that recreation is incomplete or deferred. Governance Orphaning is one route to residue, not the residue mechanism itself.

#78
Stranded Capacity Risk
RELATED · MODERATE

Both originated as cost-model blind spots in the repatriation economics work: #78 prices capacity bought and never used; #77 prices dependency the model stopped counting after elimination was declared.

#76
Repatriation Elasticity Gap
RELATED · WEAK

Shared origin in the repatriation economics model (/cloud-repatriation-cost-model/) as a separate cost variable. #77 has since been generalized beyond repatriation, so the tie is common origin, not common mechanism.

#79
Economic Persistence Bias
RELATED · WEAK

Shared origin in the repatriation economics model. #79 is the decision bias that keeps a workload in place; #77 is the dependency that remains after a workload is declared moved. Common origin, not common mechanism.

#80
Operational Amortization Window
RELATED · WEAK

Shared origin in the repatriation economics model. Unpriced residue is one reason realized savings lag the forecast inside #80's window, but #77 is no longer scoped to repatriation. Common origin, not common mechanism.

ANCHOR /dependency-residue/

FRAMEWORK HOME

Originating instance: repatriation economics (/cloud-repatriation-cost-model/). The mechanism holds for any transformation program that measures elimination against a cost model.

"A dependency isn't gone when the cost model stops counting it. It's gone when nothing still calls it, bills for it, or can be held to it."