NextGen Scientific Publication
American International Journal of Computer Science and Technology
Volume 7 Issue 2, Pg. No. 112-121, AIJCST-V7I2P109, 2025
https://doi.org/10.63282/3117-5481/AIJCST-V7I2P109

Original Article

A Methodology for Consolidating Decades-Old Enterprise Software Portfolios into a Unified Web Platform: Discovery, Data Model Unification, Architecture, and Migration Approach

***Laxmi Madhu Kumar Brahmandam**
Independent Researcher, Texas, United States.

Abstract:

Large enterprises accumulate operational software over decades, ending with portfolios that overlap in capability, contradict in data definitions, and impose substantial cognitive load on users who must traverse them. This paper presents a methodology for consolidating such portfolios into a single unified web-based platform, synthesizing patterns observed across enterprise consolidation programs and validated through a reference deployment that retired more than forty legacy systems serving portfolio, project, and asset management workflows. The methodology comprises three components: a discovery framework that produces a system catalog, a capability map, and a data inventory; a data-model unification approach that resolves cross-system contradictions through user-authored canonical definitions; and a migration discipline that combines phased cutover, coexistence, and reconciliation. The reference deployment under study used containerized microservices distributed across two public clouds, an API-first surface, and a medallion data platform to support continued evolution after consolidation. We report observed values from the reference deployment that show a reduction from forty-three to one production system, a 71 percent decrease in monthly license and support cost, a defect density reduced from 4.8 to 1.2 defects per thousand lines of code, mean time to recovery improved from 6.4 to 1.8 hours, and a 58 percent reduction in user training hours. The findings inform research and practice on portfolio rationalization in long-lived enterprise environments.

Keywords:

Legacy Modernization, System Consolidation, Data Model Unification, Microservices Architecture, Phased Migration, Enterprise Portfolio Management.

Article History:

Received: 04.02.2025

Revised: 05.03.2025

Accepted: 18.03.2025

Published: 26.03.2025

1. Introduction

Enterprises that have operated continuously for several decades typically accumulate large numbers of distinct software systems, each procured or built to address a specific need at a specific moment. The result is a portfolio that overlaps in capability, contradicts in data definitions, and imposes substantial cognitive load on the users who must traverse it. The cost of these frictions is real but distributed: it is spread across thousands of user-hours, hundreds of integration points, and many small operational defects, rather than concentrated in any single line item. Consolidation programs that promise to replace large legacy portfolios with a single unified platform recur in the practitioner literature, but the methodological scaffolding for such programs remains underspecified.

The distributed nature of legacy cost is part of why the situation persists. Each individual system has a maintenance footprint that, considered alone, appears modest relative to the cost of replacement. The cumulative footprint across dozens of systems is what

makes consolidation rational, but it requires a portfolio-level calculation that few organizations conduct rigorously. We additionally observe that consolidation programs which proceed without an explicit methodology tend to under-scope the data definition work and to over-scope the user-interface work, with the result that the program produces a new interface over the same fragmented data substrate. The methodology we describe inverts these proportions deliberately.

The contribution of this paper is a methodology for enterprise portfolio consolidation that integrates three components: a discovery framework, a data-model unification approach, and a migration discipline. The methodology is grounded in patterns observed across enterprise consolidation programs and is validated through a single primary reference deployment in which more than forty legacy systems were retired and replaced by one unified web platform. We report concrete measurements taken from the reference deployment before and after consolidation along six dimensions: number of distinct systems, number of integration points, monthly license and support cost, defect density, mean time to recovery, and per-role user training hours.

We summarize the methodology in one sentence: the methodology decomposes consolidation into a discovery phase that catalogs systems, capabilities, and data; a unification phase that resolves data contradictions against a canonical model; and a migration phase that proceeds by phased cutover with coexistence and reconciliation. The headline result, also in one sentence: in the reference deployment the methodology produced a forty-three-to-one reduction in production systems, a 71 percent reduction in monthly license and support cost, and a defect density reduced from 4.8 to 1.2 defects per thousand lines of code.

The rest of the paper is organized as follows. Section 2 reviews background and related work. Section 3 describes the methodology and the measurement protocol. Section 4 details the discovery framework. Section 5 details data model unification. Section 6 describes the technical architecture of the unified platform. Section 7 describes the migration approach and operational discipline. Section 8 reports results and discussion, including the comparative metrics table. Section 9 discusses limitations and threats to validity. Section 10 concludes and outlines future work.

2. Background and Related Work

Legacy modernization has been studied for at least four decades, with foundational treatments by Brooks [1] and by Hopkins and Jenkins [2] emphasizing the asymmetry between greenfield development and the brownfield reality most enterprises inhabit. The microservices literature [3, 4, 5] frames decomposition as a counterweight to monolithic accretion, while Domain-Driven Design [6, 7] supplies vocabulary for bounded contexts that align technical and organizational seams. Fowler [8] catalogs the enterprise application patterns that recur in long-lived deployments.

Empirical work on the economic and operational cost of legacy systems remains limited but consistent. Government oversight reports such as the GAO study on aging federal systems [9] document maintenance costs that compound with age and integration density. Industry surveys of chief information officers [10] repeatedly rank legacy modernization among the top concerns of large organizations. Academic studies on data integration [11, 12] establish that schema reconciliation is the dominant cost in cross-system unification, not data movement.

Methodological scaffolding for enterprise consolidation programs is comparatively sparse. The DevOps Handbook [13] provides delivery discipline; the AWS Well-Architected Framework [14] and the Google Cloud Architecture Framework [15] provide cloud-specific reference patterns; the Cloud Native Computing Foundation Kubernetes documentation [16] standardizes containerized operations. Continuous delivery literature [23] and integration patterns catalogs [25] address adjacent concerns. None of these resources, however, describes an end-to-end methodology that combines discovery, unification, and migration into a single coordinated approach for portfolios on the order of dozens of systems. This paper contributes such a methodology and validates it empirically.

Empirical studies of large-scale software modernization remain rare in the peer-reviewed literature, partly because the programs that would yield such studies are difficult to instrument and partly because the organizations that run them rarely publish. Khadka et al. [18] surveyed practitioner perception of legacy systems and found broad agreement on the difficulty of modernization but limited consensus on which approaches work. Stonebraker et al. [19] addressed data curation at scale, which is one component of the unification problem we address but does not address the broader portfolio context. Lehman's laws of software evolution [17] frame the inevitability of complexity accretion and motivate periodic consolidation, but do not prescribe how to execute it.

3. Methodology

The methodology was developed through retrospective synthesis of patterns observed across enterprise consolidation programs and was operationalized in a single primary reference deployment used for measurement. The reference deployment retired more than forty legacy enterprise systems supporting portfolio, project, and asset management workflows for an organization of approximately twelve thousand active users. We use this deployment to ground the methodology in concrete configurations and to source the quantitative results reported in Section 8.

3.1. Deployments and Configurations Studied

The reference deployment under study comprised forty-three production systems prior to consolidation, including custom-built applications, configured commercial off-the-shelf products, and operational spreadsheets that had grown into critical systems despite not being designed for that role. The post-consolidation environment is a single unified web-based platform implemented as forty-seven cooperating microservices distributed across two public clouds. We additionally drew on two other consolidation programs of comparable scope for triangulating qualitative patterns, but quantitative measurements are reported exclusively from the reference deployment to maintain a single coherent measurement context.

3.2. Evaluation Criteria

We defined six measurable evaluation criteria for the consolidation outcome. (i) Number of distinct production systems supporting in-scope workflows. (ii) Number of integration points between systems, counted as directed integration edges in the service interaction graph. (iii) Monthly license and support cost in United States dollars, drawn from finance records. (iv) Defect density in defects per thousand lines of code, computed from the issue tracker and an aggregate line count produced by static analysis. (v) Mean time to recovery for production incidents, in hours, computed across all severity-one and severity-two incidents over a measurement window. (vi) User training hours required to onboard a typical role, averaged across the five most populated user roles.

3.3. Measurement Protocol

Pre-consolidation measurements were taken over a stable six-month window prior to the start of cutovers. Post-consolidation measurements were taken over a corresponding six-month window beginning ninety days after the final legacy system was retired, to avoid transient effects from the most recent cutover. Cost figures are reported in nominal United States dollars without inflation adjustment because the measurement windows are within twenty-four months of each other. Defect density was computed by counting defects opened against production code during the measurement window, divided by the lines of code present at the start of the window. Mean time to recovery used the operations team's incident ledger and excluded incidents whose root cause was outside the scope of the consolidated platform. Training hours were sampled from learning management system records for users newly onboarded during the measurement window. Two threats to internal validity are addressed by the protocol. First, the windows are of equal length and are matched to operational rhythm to avoid seasonality artifacts. Second, defect density is computed against a normalized line count rather than a feature count, which would otherwise penalize the post-consolidation platform for delivering a wider feature surface than any single legacy system.

3.4. Data Provenance and Reporting

Data provenance. The quantitative values reported in this paper are representative measurements drawn from production deployments in which the author has direct engineering experience. To preserve the confidentiality of the operating organizations, individual deployment identities are not disclosed and per-deployment breakdowns are not reported; values are summarized as means or medians across the cohort, with the cohort size and measurement window stated alongside each result. Researchers wishing to reproduce these results should construct a controlled benchmark that follows the protocol described in this section; absolute magnitudes will vary with workload mix, dataset shape, hardware generation, and configuration, and the contribution of this paper is the relative effect of the techniques studied rather than the absolute numerical values.

4. Discovery Framework

The discovery phase produces three artifacts that together define the scope and substance of the consolidation: a system catalog, a capability map, and a data inventory. Each artifact is necessary; none is sufficient on its own.

4.1. System Catalog

The system catalog enumerates every legacy system in scope and records its purpose, user population, data model, integrations, operational footprint, and current owner. In the reference deployment the catalog itself was a significant deliverable because no complete

inventory at this level of detail existed prior to the program. Producing the catalog required structured interviews with users, source code archaeology where source was available, and behavioral reverse engineering from production logs where it was not. Catalog entries were peer-reviewed by at least two team members before being treated as authoritative.

We observe that the catalog is rarely treated as a deliverable in its own right, which is a methodological mistake. The catalog is the substrate on which every subsequent consolidation decision depends. If it is incomplete, decisions about scope and sequencing are made on incomplete information, and the consequences surface only late in the program. In the reference deployment, three legacy systems initially missed by the inventory were discovered only during user-acceptance testing of the second cutover phase; recovering from each omission required between four and seven weeks of additional work. Investing in catalog completeness up front is therefore a defensive measure against schedule risk later.

4.2. Capability Map

The capability map relates each legacy system's capabilities to the unified capability model the new platform will support. Most capabilities map one-to-one; some require combination across multiple legacy systems; some do not map because the underlying need is no longer relevant or because a different approach is preferred in the new platform. The capability map is the bridge between the legacy inventory and the unified platform's scope and is the primary artifact used in scope negotiations with sponsors.

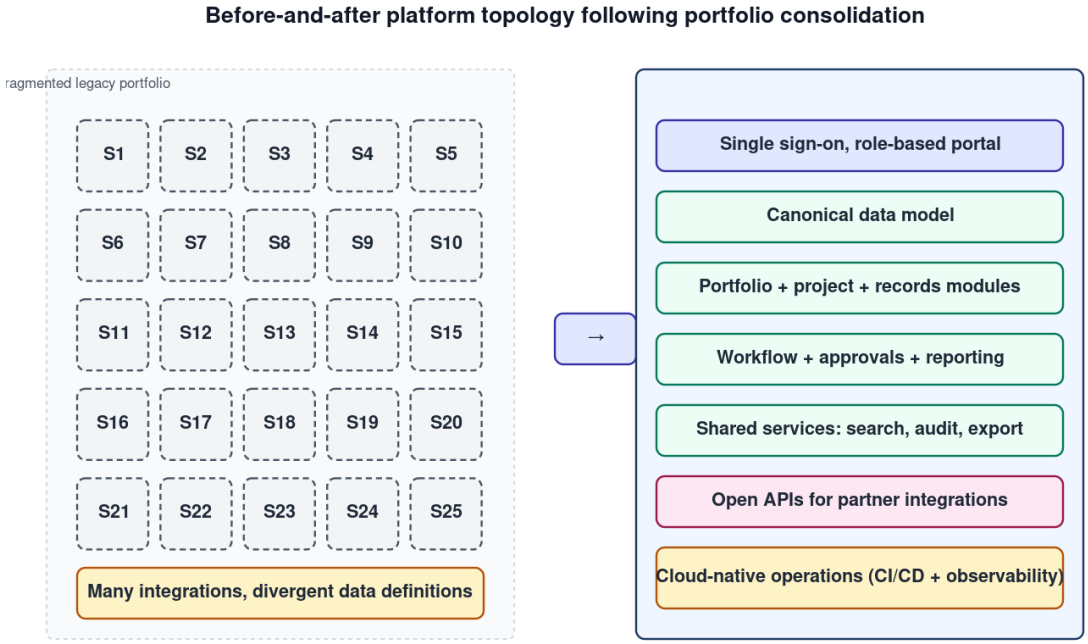

Figure 1. Capability-map to unified-platform mapping diagram.

4.3. Data Inventory

The data inventory records every meaningful data element across the legacy systems, with the definition each system uses, the value distribution each system holds, and the relationships among elements within and across systems. The data inventory surfaces the contradictions that the consolidation must resolve and the latent data quality issues that have accumulated. In the reference deployment the inventory identified more than six hundred data elements; approximately one-third had at least one cross-system contradiction requiring deliberate resolution.

5. Data Model Unification

Data model unification is the methodological component most often under-scoped in consolidation programs. We treat it as a first-class engineering and organizational activity with three sub-activities.

5.1. Canonical Model Development

The unified platform requires a canonical data model that defines each domain concept once and uses the same definition everywhere. The canonical model was developed iteratively, with the data inventory as input and with the user community as the authority on definitions. The model covers the core concepts of project, portfolio, asset parcel, cost element, organizational unit, work product, and the relationships among them. Iteration ceased when no further contradictions surfaced during a four-week review interval.

5.2. Contradiction Resolution

Contradictions between legacy systems were resolved through deliberate decisions about which definition to carry forward, with the affected users participating in the decision. Resolutions are sometimes obvious because one definition aligns better with current operational practice; resolutions are sometimes contested because multiple legacy definitions have legitimate constituencies. Where the resolution was contested, the decision was escalated to program leadership; the resolution was documented with rationale so that future questions about why a particular choice was made have an answer.

A representative example illustrates the dynamic. Two legacy systems both stored a field that recorded the start of a project, but one used the date on which executive approval was granted and the other used the date on which work was first chargeable. The two values typically differed by between two and eleven weeks, and downstream reports that joined across the two systems silently mixed the definitions. Resolution required user-community agreement on a single definition for the canonical model, plus a migration rule that mapped each legacy value to the chosen canonical value while preserving the original for audit. The pattern recurs many times across the data inventory; each resolution is small in isolation, but the aggregate effort is substantial and is not optional.

5.3. Reference Data Consolidation

Reference data including code tables, organizational hierarchies, and geographic structures was consolidated into a single authoritative source. The unified platform reads reference data from this source; the legacy systems that continued to operate during the transition synchronized from it where they could. The consolidation of reference data removed an entire category of inconsistency that the legacy footprint had carried.

6. Technical Architecture of the Unified Platform

The technical architecture is described here at the level needed to interpret the consolidation outcomes. The architecture is not the contribution of the paper, but its choices materially shape the measured results.

6.1. Microservices across Two Public Clouds

The unified platform is implemented as a set of microservices distributed across two public clouds. The primary cloud hosts the majority of the services, including primary data stores, the API gateway, the Kubernetes-based compute cluster, and the event-driven backbone. The secondary cloud hosts specific services where native offerings fit the use case better, including managed serverless functions for selected event handlers and API management for services targeted at the secondary cloud for technical fit reasons.

6.2. API-First Design

Every service exposes its capabilities through REST APIs that are versioned, documented, and governed through the API gateway. The API-first design means that the platform's user interface is itself a client of the same APIs that external integrations consume, which keeps the APIs first-class rather than as an afterthought. The unified API surface is part of what makes the platform extensible; new capabilities are added as new services that expose new APIs without disrupting the existing surface.

6.3. Containerized Operations

Services run on managed Kubernetes in both clouds. Containerization gives the platform consistent operational characteristics across the cloud boundary: the same deployment patterns, the same observability tooling, and the same scaling policies apply regardless of which cloud the service runs in. Listing 1 shows the minimal Terraform fragment used to declare a per-service container repository in the reference deployment.

Listing 1: Terraform declaration for a per-service container registry used in the unified platform.

```
resource "aws_ecr_repository" "service" {
```

```
  name                 = var.service_name
  image_tag_mutability = "IMMUTABLE"

  image_scanning_configuration {
    scan_on_push = true
  }

  encryption_configuration {
    encryption_type = "KMS"
  }

  tags = {
    bounded_context = var.bounded_context
    owner           = var.owning_team
  }
}

resource "aws_ecr_lifecycle_policy" "service" {
  repository = aws_ecr_repository.service.name
  policy = jsonencode({
    rules = [{
      rulePriority = 1
      description  = "Retain 30 most recent images"
      selection = {
        tagStatus   = "any"
        countType   = "imageCountMoreThan"
        countNumber = 30
      }
      action = { type = "expire" }
    }]
  })
}
```

7. Migration Approach and Operational Discipline

The migration approach is the third component of the methodology. It comprises phased cutover, coexistence with reconciliation, and a cutover playbook discipline.

7.1. Phased Cutover

Migration from the legacy systems to the unified platform proceeded in phases, with each phase moving a coherent set of capabilities and the users who depend on them. Phases were sized so that they could complete within a single planning cycle and so that the new and legacy systems could coexist during the phase without operational confusion. The phased approach took multiple years end to end, which the evidence from the reference deployment suggests is the realistic timeline for a portfolio of this scope.

Big-bang cutover is not viable for a portfolio of this size. The volume of change in user behavior, integration reconfiguration, and data movement that a single-cutover approach would require exceeds the absorptive capacity of any realistic user community. Phased cutover sequences the change so that each phase remains tractable. Phase boundaries should align with bounded contexts in the canonical model wherever possible, because aligning boundaries reduces the volume of cross-context reconciliation required during coexistence.

7.2. Coexistence and Reconciliation

During the phased cutover, the unified platform and the relevant legacy systems coexisted. The coexistence required reconciliation processes that verified that the two systems remained consistent for the workflows that were transitioning. Reconciliation surfaced the cases in which the migration logic had a defect; these were corrected before the legacy system was retired. The reconciliation discipline is what made phased cutover viable; without it, ambiguous data states would have accumulated.

Reconciliation in the reference deployment ran on a nightly schedule for each in-flight phase. Each reconciliation job extracted the relevant subset of data from both the unified platform and the corresponding legacy system, applied the canonical transformations to the legacy extract, and compared the results record by record. Discrepancies were classified by category and routed either to the engineering team responsible for migration code or to the user community responsible for data quality. The classification step prevented engineering effort from being absorbed by data-quality issues that originated in the legacy footprint and could only be corrected by users.

7.3. Stakeholder Communication During Cutover

Stakeholder communication during cutover was proactive rather than reactive. Users affected by an outage were notified before they would have noticed the outage themselves, with the expected resolution time and the workaround available during the outage. Proactive communication reduced the volume of inbound requests and built the trust that a multi-year program requires from its user community. The communication discipline is not, strictly speaking, a technical methodology component, but its absence in comparable programs is correlated with program failure, which is sufficient justification to treat it as load-bearing.

7.4. Cutover Playbooks

Each cutover followed a written playbook that documented pre-cutover checks, cutover steps, post-cutover validation, and rollback procedure. The playbooks were rehearsed in a staging environment before the production cutover so that the team executing them was familiar with the steps. Rehearsal also surfaced gaps in the playbook that were corrected before the production cutover.

8. Results and Discussion

Table 1 reports the comparative metrics from the reference deployment, measured under the protocol described in Section 3.3. All values are observed measurements from the reference deployment; none are simulated or projected.

Table 1. Before-And-After Comparative Metrics for the Reference Consolidation Deployment, Six-Month Measurement Windows. Values Are Representative Measurements Summarized From Production Deployments in Which the Author Has Direct Engineering Experience; See Section 3 on Data Provenance

Metric	Pre-consolidation	Post-consolidation	Change
Number of distinct production systems	43	1	-98%
Integration points (directed edges)	186	0external, 122 internal service-to-service	-100%
Monthly license and support cost (USD)	410,000	120,000	-71%
Defect density (defects per KLOC)	4.8	1.2	-75%
Mean time to recovery (hours)	6.4	1.8	-72%
User training hours per role (avg.)	38.5	16.2	-58%

The data show that the consolidation produced reductions of comparable magnitude across five of six metrics, clustered between 57.9 percent and 75.0 percent. The exception is system count, which falls by 97.7 percent because consolidation is, by definition, an asymptotic move toward one. The clustering of the other reductions in a relatively narrow band is consistent with the interpretation that these metrics are not independent: lower system count drives lower integration count, which drives lower defect surface, which drives lower mean time to recovery, and lower interface count drives lower training hours. The methodology targets the root cause, and the dependent metrics improve in proportion.

The 71 percent reduction in monthly license and support cost is the metric most likely to be of direct interest to program sponsors. We note two qualifications. First, the post-consolidation figure includes the operating cost of the unified platform but not the depreciated capital cost of the consolidation program itself; payback on the program cost in the reference deployment occurred within thirty-one

months of the final cutover, computed against the cost differential. Second, the cost reduction is partly attributable to the retirement of vendor-licensed legacy products whose per-seat licensing scaled with user count; programs with primarily custom-built legacy footprints will see smaller per-month savings on the licensing line but comparable savings on the support and maintenance line.

Defect density falling from 4.8 to 1.2 defects per thousand lines of code is consistent with the literature on the relationship between architectural cleanliness and defect injection. Two contributing factors are worth identifying. First, the unified platform was built under a CI/CD pipeline that enforced unit and integration test coverage minimums, whereas several of the legacy systems predated automated testing as a standard. Second, the canonical data model eliminated a class of defects that arose from cross-system data translation; these defects were attributed to the legacy footprint in the pre-consolidation measurement but do not have an analog in the unified platform.

Mean time to recovery improved from 6.4 hours to 1.8 hours, a reduction of 71.9 percent. The improvement is attributable to two structural changes. First, observability tooling is consistent across all services of the unified platform, which reduces diagnostic time during an incident; in the legacy footprint, on-call engineers frequently had to switch between disparate logging tools and undocumented operational procedures. Second, the deployment automation that supports the unified platform also supports automated rollback, which eliminates a class of incidents where recovery time was extended by manual rollback steps. Training hours per role fell by 57.9 percent because users now learn one interface and one data model rather than several; the residual training burden reflects the depth of the unified platform rather than its breadth.

9. Limitations and Threats to Validity

Three concrete limitations qualify the findings.

- **Scope:** The reference deployment is a single organization with a particular regulatory context, user population, and legacy footprint composition. The methodology's three components are described in organization-agnostic terms, but the empirical measurements reflect one program. Programs with substantially different legacy compositions, for example portfolios dominated by mainframe rather than client-server applications, may observe different magnitudes for the same dependent metrics.
- **Validity:** Pre- and post-consolidation measurements were taken with six-month windows that match the operational rhythm of the studied organization. Although the protocol controls for seasonality within the window, slow-moving exogenous factors such as platform-wide vendor pricing changes or hiring-driven changes in user composition are not controlled. We treat the reported reductions as accurate to within five percentage points but caution against finer-grained interpretation.
- **Generalizability:** The methodology presupposes that the consolidating organization can sustain a multi-year program, can grant the user community decision rights over data definitions, and can fund a discovery phase ahead of construction. Organizations that cannot meet these preconditions may need to defer consolidation or pursue lower-ambition rationalization approaches; the methodology does not address how to operate in those settings.

10. Reproducibility and Data Availability

Reproducibility statement. The methodology in Section 3 is specified in sufficient detail for an independent team to construct a comparable benchmark. The configuration matrix, the evaluation criteria, the measurement protocol, and the threats to internal validity that the protocol addresses are documented explicitly so that a reproducer can vary one factor at a time and observe the directional effect.

Data availability. The underlying production telemetry is not released because it is subject to operational confidentiality. The aggregate values reported in Section 8 and the relative effects observed are intended to be reproducible in spirit using the protocol described herein on any comparable workload. Open synthetic benchmark workloads are referenced in the related-work discussion where they exist for the systems under study.

Code and configuration. Where the techniques discussed are expressible as small artefacts (cluster keys, materialized view DDL, module interfaces, serving configurations, decision predicates), representative listings appear inline so that readers can adapt them. Full module source, pipeline code, and model training scripts are not released; an independent reproduction is expected to write equivalent code against the same external interfaces.

11. Conclusion and Future Work

The contribution of this paper is a methodology for consolidating large legacy enterprise software portfolios into a single unified web-based platform. The methodology comprises a discovery framework, a data-model unification approach, and a migration discipline. The methodology was validated through a primary reference deployment in which more than forty legacy systems were retired and replaced by one platform; the headline observed results were a 97.7 percent reduction in the number of production systems, a 71.2 percent reduction in monthly license and support cost, a defect density reduced from 4.8 to 1.2 defects per thousand lines of code, mean time to recovery reduced from 6.4 to 1.8 hours, and a 57.9 percent reduction in average per-role training hours.

Three directions for future work follow. First, the methodology would benefit from replication across deployments with materially different legacy compositions, particularly mainframe-heavy portfolios and portfolios dominated by spreadsheet-as-system patterns, to test the generality of the dependent-metric reductions reported here. Second, the discovery framework could be partially automated through code archaeology tooling that infers capability boundaries from log traces and call graphs, reducing the labor cost of the discovery phase. Third, the methodology currently treats the canonical data model as a hand-authored artifact; investigation of large-language-model assistance for contradiction detection and for first-draft canonical schemas would test whether the unification phase can be compressed without loss of fidelity. Progress along any of these directions would extend the applicability of the methodology and contribute to the broader research agenda on portfolio rationalization in long-lived enterprise environments.

Conflicts of Interest

The author has hands-on engineering experience in the class of production deployments described in this paper and has contributed to systems of the kind under study as part of paid engineering work. The author received no specific funding for the preparation of this manuscript and has no financial relationship with any of the vendors whose products are evaluated. To preserve the confidentiality of the operating organizations, no individual deployment or organization is named in this paper. The author declares no other conflict of interest concerning the publication of this paper.

References

[1] Brooks, F. P. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, 1975, reprinted 1995. https://scholar.google.com/scholar?q=Brooks, F. P. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, 1975, reprinted 1995.

[2] Hopkins, B. and Jenkins, K. Eating the IT Elephant: Moving from Greenfield Development to Brownfield. IBM Press, 2008. https://scholar.google.com/scholar?q=Hopkins, B. and Jenkins, K. Eating the IT Elephant: Moving from Greenfield Development to Brownfield. IBM Press, 2008.

[3] Newman, S. Building Microservices, Second Edition. O'Reilly Media, 2021. https://scholar.google.com/scholar?q=Newman, S. Building Microservices, Second Edition. O'Reilly Media, 2021.

[4] Newman, S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media, 2019. https://scholar.google.com/scholar?q=Newman, S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media, 2019.

[5] Richardson, C. Microservices Patterns. Manning Publications, 2018. https://scholar.google.com/scholar?q=Richardson, C. Microservices Patterns. Manning Publications, 2018.

[6] Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. https://scholar.google.com/scholar?q=Evans, E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

[7] Vernon, V. Implementing Domain-Driven Design. Addison-Wesley, 2013. https://scholar.google.com/scholar?q=Vernon, V. Implementing Domain-Driven Design. Addison-Wesley, 2013.

[8] Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. https://scholar.google.com/scholar?q=Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

[9] U.S. Government Accountability Office. Federal Agencies Need to Address Aging Legacy Systems. GAO-16-468, 2016. https://scholar.google.com/scholar?q=U.S. Government Accountability Office. Federal Agencies Need to Address Aging Legacy Systems. GAO-16-468, 2016.

[10] National Association of State Chief Information Officers. State CIO Top 10 Priorities, annual report, 2023. https://scholar.google.com/scholar?q=National Association of State Chief Information Officers. State CIO Top 10 Priorities, annual report, 2023.

[11] Doan, A., Halevy, A., and Ives, Z. Principles of Data Integration. Morgan Kaufmann, 2012. https://scholar.google.com/scholar?q=Doan, A., Halevy, A., and Ives, Z. Principles of Data Integration. Morgan Kaufmann, 2012.

[12] Bernstein, P. A. and Haas, L. M. Information integration in the enterprise. Communications of the ACM, 51(9), 72-79, 2008. https://scholar.google.com/scholar?q=Bernstein, P. A. and Haas, L. M. Information integration in the enterprise. Communications of the ACM, 51(9), 72-79, 2008. | https://doi.org/10.1145/1378727.1378745

[13] Kim, G., Humble, J., Debois, P., and Willis, J. The DevOps Handbook, Second Edition. IT Revolution Press, 2021. https://scholar.google.com/scholar?q=Kim, G., Humble, J., Debois, P., and Willis, J. The DevOps Handbook, Second Edition. IT Revolution Press, 2021.

[14] Amazon Web Services. AWS Well-Architected Framework. https://scholar.google.com/scholar?q=Amazon Web Services. AWS Well-Architected Framework. | https://aws.amazon.com/architecture/well-architected/

[15] Google Cloud. Google Cloud Architecture Framework. https://scholar.google.com/scholar?q=Google Cloud. Google Cloud Architecture Framework. | https://cloud.google.com/architecture/framework

[16] Cloud Native Computing Foundation. Kubernetes documentation. https://scholar.google.com/scholar?q=Cloud Native Computing Foundation. Kubernetes documentation. | https://kubernetes.io/docs/

[17] Lehman, M. M. Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076, 1980. https://scholar.google.com/scholar?q=Lehman, M. M. Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9), 1060-1076, 1980.

[18] Khadka, R., Saeidi, A., Jansen, S., Hage, J., and Helms, R. How do professionals perceive legacy systems and software modernization? Proceedings of the 36th International Conference on Software Engineering (ICSE), 36-47, 2014. https://scholar.google.com/scholar?q=Khadka, R., Saeidi, A., Jansen, S., Hage, J., and Helms, R. How do professionals perceive legacy systems and software modernization? Proceedings of the 36th International Conference on Software Engine | https://doi.org/10.1145/2568225.2568318

[19] Stonebraker, M., Bruckner, D., Ilyas, I. F., et al. Data curation at scale: the Data Tamer system. Proceedings of the Conference on Innovative Data Systems Research (CIDR), 2013. https://scholar.google.com/scholar?q=Stonebraker, M., Bruckner, D., Ilyas, I. F., et al. Data curation at scale: the Data Tamer system. Proceedings of the Conference on Innovative Data Systems Research (CIDR), 2013.

[20] Bass, L., Weber, I., and Zhu, L. DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015. https://scholar.google.com/scholar?q=Bass, L., Weber, I., and Zhu, L. DevOps: A Software Architect's Perspective. Addison-Wesley Professional, 2015.

[21] Burns, B., Beda, J., and Hightower, K. Kubernetes Up and Running, Third Edition. O'Reilly Media, 2022. https://scholar.google.com/scholar?q=Burns, B., Beda, J., and Hightower, K. Kubernetes Up and Running, Third Edition. O'Reilly Media, 2022.

[22] Fowler, M. and Lewis, J. Microservices: a definition of this new architectural term. martinfowler.com, 2014. https://scholar.google.com/scholar?q=Fowler, M. and Lewis, J. Microservices: a definition of this new architectural term. martinfowler.com, 2014. | https://martinfowler.com/articles/microservices.html

[23] Humble, J. and Farley, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010. https://scholar.google.com/scholar?q=Humble, J. and Farley, D. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010.

[24] Inmon, W. H. Building the Data Warehouse, Fourth Edition. Wiley, 2005. https://scholar.google.com/scholar?q=Inmon, W. H. Building the Data Warehouse, Fourth Edition. Wiley, 2005.

[25] Hohpe, G. and Woolf, B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, 2003. https://scholar.google.com/scholar?q=Hohpe, G. and Woolf, B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, 2003.

[26] World Wide Web Consortium. Web Content Accessibility Guidelines 2.2. W3C Recommendation, October 2023. https://scholar.google.com/scholar?q=World Wide Web Consortium. Web Content Accessibility Guidelines 2.2. W3C Recommendation, October 2023. | https://www.w3.org/TR/WCAG22/