International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(23s) 2026 pp. 1614-1624
https://doi.org/10.70917/ijcisim-2026-5716
Article

Content Platform Modernization Across Insurance and Financial Services Digital Estates

Maulik Darji

Independent Researcher, India
Email:darjimaulik@gmail.com

Abstract: Digital estates in insurance and financial services tend to accumulate architecture rather than choose it, and after enough product launches and regulatory changes the result is difficult to maintain and slow to publish through. This paper reports the modernization of one such estate, moving from a monolithic architecture in which authored content was embedded directly in presentation elements to a headless, component-based platform serving multiple channels. The programme covered roughly 3,000 pages, just over 60% of the estate, on platforms reaching a customer base of approximately seven million. Three things are reported. The first is a sequencing strategy that kept the production estate live throughout, combining dual-track environments, incremental feature isolation, load-balancer switching with automated failback, and a content parity check that withheld each cutover until source and target agreed. The second is an explicit rebuild-versus-migrate decision framework scored on feasibility, capability gap and delivery impact, which produced a rebuild decision for four of eight subsystems and a migrate-and-refactor decision for the remainder. The third is the treatment of editorial governance as a migration invariant: approval workflow state, publishing permission boundaries and audit logging were carried across the platform change rather than reconstructed after it, because records obligations in this sector do not pause for a re-platforming. The website's performance improved due to the headless architecture. This is a single-organisation field report rather than a controlled study, and the paper is explicit throughout about which of its accounts are reported by the programme rather than measured under a controlled protocol.
Keywords: content management, legacy modernization, headless architecture, strangler fig, regulated industries, developer experience

1. INTRODUCTION

Large digital estates in insurance and financial services are rarely designed as a whole. They are added to. Each product launch, regulatory change and rebrand leaves another layer behind it, and after a decade the platform underneath carries a great many decisions whose original reasoning has left the organisation along with the people who made them. The estate described here had reached that state. It ran on monolithic, tightly coupled architecture built over legacy .NET and ASP.NET foundations, on older content management versions, with custom sublayout structures in which authored content sat directly inside presentation elements rather than being managed independently of them.The cost of this arrangement was not primarily an engineering cost. Engineering teams adapt to bad architecture, and this one had. The cost fell on the people who publish. Authors waited on build and page times that had grown with the estate. Search indexing was slow enough that reindexing competed with publishing for the same server capacity, and publishing cycles were resource-heavy enough to be scheduled around rather than run on demand. Feature rollout was delayed because every change touched presentation and content together. Personalising a user journey safely was difficult, because there was no clean place to vary content without varying markup.

Migration research has established that these motivations are typical. Di Francesco et al. [3] surveyed industrial practitioners and found maintainability and delivery speed dominating the decision to migrate, ahead of any purely technical consideration. What the literature has not established is what to do when the thing being migrated is a content platform rather than a transactional backend.

That distinction matters more than it first appears. Code sits in version control, where it can be branched and diffed and merged, and where the history of any given line is recoverable. Authored content does not behave that way. It carries editorial state, it keeps changing while the migration is underway, it belongs to people outside engineering who have their own deadlines, and in a regulated sector it carries approval history and retention obligations that are themselves subject to inspection. SEC Rule 17a-4 [14] requires that electronic records preserve the original record, maintain an index, and remain available for regulatory production. An item that arrives on the new platform with its approval state regressed is not a content bug that can be fixed on Monday. The record itself is now wrong.

This paper reports how one such migration was sequenced and what it produced. The contributions are:

1. A cutover strategy for content estates that keeps production live throughout, in which a content parity check acts as the gate on each incremental switch.

2. An explicit rebuild-versus-migrate decision framework, applied across eight subsystems, with the reasoning recorded rather than implied.

3. A treatment of editorial governance and audit obligation as invariants to be carried across the migration rather than restored afterwards.

Section 2 positions this against the modernization literature. Section 3 describes the estate before the programme. Sections 4 through 6 cover the migration strategy, the decision framework and governance continuity. Section 7 records what took longer than expected, and Section 8 states the limitations, which are substantial and are not softened.

All organisational identifiers have been generalised. Vendor and product names are retained where they are load-bearing to the technical account.

2. Background and Related Work

Three strands of literature bear on this work, and the gap sits between them rather than inside any one.

2.1 Modernization and migration

Modernization is among the best-served topics in empirical software engineering. Martínez Saucedo et al. [1] provide the most current systematic map of monolith-to-microservice migration, and Abgaz et al. [2] a Q1 systematic review of decomposition approaches and the criteria used to select them. Bogner et al. [4] contribute the industrial view of what happens after migration, when the decomposed system has to stay maintainable. Balalaie et al. [6] remains the canonical industrial migration experience report and is the closest precedent in form to this paper.

What all of this work has in common is that it concerns backends. Service boundary identification, data ownership, transactional consistency, deployment topology: these are the recurring problems, and they are well covered. The layer where content is authored, approved and delivered barely appears.

Assunção et al. [11] make an observation that motivates this paper directly: the sheer diversity of available modernization strategies can itself misguide practitioners looking for guidance fitted to their situation. A practitioner facing a content estate has an abundance of strategy and an absence of anything situated.

2.2 Presentation and content layers

The closest adjacent work is on micro-frontends. Peltonen et al. [7] conducted a multivocal literature review precisely because the peer-reviewed evidence base was too thin to review on its own, and had to admit grey literature to reach a usable corpus. That admission is the strongest available evidence for the gap this paper occupies. If presentation-layer decomposition is under-researched, content platform modernization, which sits above it and adds editorial workflow, is barely researched at all.

2.3 Developer experience

Forsgren et al. [9] established with the SPACE framework that developer productivity cannot honestly be reduced to a single metric, and that any credible measurement covers multiple dimensions including at least one perceptual measure. Noda et al. [10] narrowed the target with the DevEx framework, identifying feedback loops, cognitive load and flow state as the primary drivers, and cognitive-load reduction as the highest-leverage intervention available.

Both are framework contributions. Reports of a specific inner-loop intervention introduced during a migration, described in those terms, remain rare, and the measurement standard SPACE sets is part of the reason.

2.4 The gap

Taken together: the field knows how to decompose a backend, knows that presentation-layer decomposition is under-evidenced, and knows how to think about developer experience in the abstract. It does not have an account of modernizing a content platform in a regulated sector, where the approval regime is as much a migration constraint as the data schema.

Related Work in this paper is therefore built on adjacent literature rather than direct predecessors. That is stated plainly here rather than disguised through selective citation.

3. The Estate Before Modernization

The legacy platform was a monolithic, tightly coupled architecture. Presentation was implemented as custom sublayout structures on legacy ASP.NET and WebForms foundations. Content was not managed as an independent resource: it was embedded directly into presentation elements, so that changing what a page said and changing how a page rendered were the same operation, performed by the same people, deployed on the same cycle.

Figure 1 contrasts this arrangement with the target. The salient feature of the legacy side is the absence of a seam. There is no point at which content can be addressed independently of the markup that displays it, which is why omnichannel delivery was not a matter of building another consumer but of rebuilding the estate.

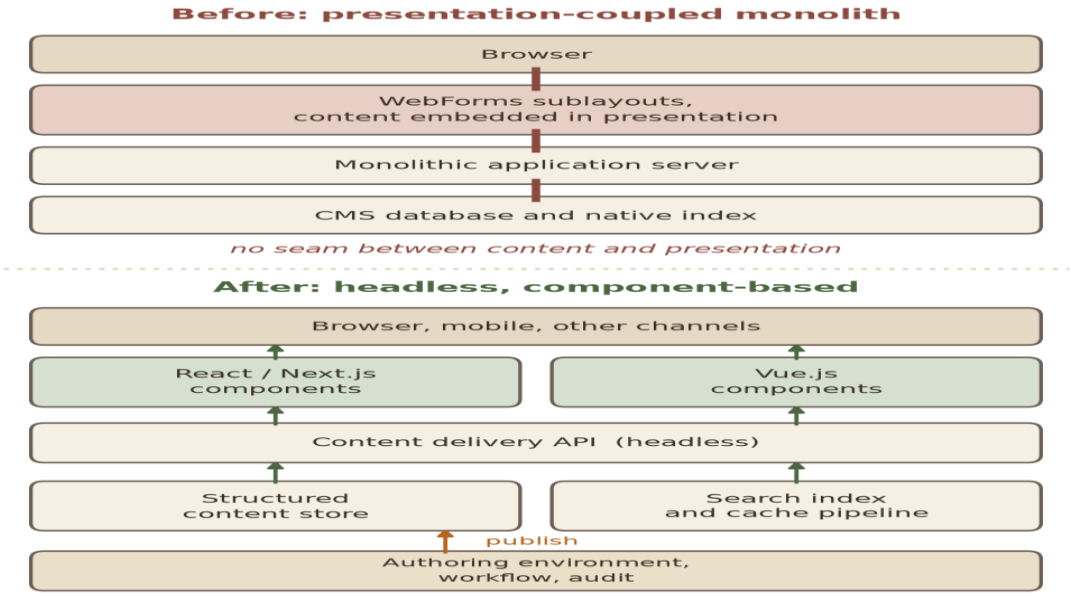

Figure 1. Estate architecture before and after modernization. The legacy stack has no seam between content and presentation; the modernized stack introduces a content delivery API that permits multiple consumers over one structured content store.

Table 1 summarises the characteristics that drove the programme.

Table 1. Estate characteristics before and after modernization.

Dimension	Before	After
Presentation	ASP.NET and WebForms sublayouts	React, Next.js and Vue.js components
Content coupling	Embedded in presentation elements	Managed independently, delivered by API
Search	CMS-native index, rebuild competes with publishing	Dedicated search service with auto-updating index pipeline and custom caching
Authoring cost	High build and page times for content authors	Componentized authoring, no rebuild for content change
Publishing	Resource-heavy cycles loading CMS servers	Scheduled independently of delivery load
Channel reach	Web only, per-channel rebuild required	Multiple channels over one content API
Personalisation	Difficult to vary content without varying markup	Content variation independent of presentation
Batch operations	Ad hoc scripting, per-item calls	Bulk service path, batched updates in a single run

The technical debt was not confined to any one layer, and no single defect explains the state. It accumulated across years of infrastructure that was individually reasonable and collectively unmaintainable, which is the ordinary form of the problem rather than an unusual one.

4. Migration Strategy

The constraint that shaped everything: the estate could not go down. Insurance and financial services digital properties carry transactional and disclosure functions, and a maintenance window long enough for a cutover of this size did not exist.

Four mechanisms together made incremental replacement possible.

4.1 Dual-track environments

The legacy production site was kept fully operational on its existing environment while the modernized architecture was stood up in parallel. Development and validation happened entirely on the second track. Live traffic was never exposed to a partially migrated state, and at no point did the programme depend on the new platform being ready in order for the business to keep running.

4.2 Incremental feature isolation

Rather than a single cutover, individual features were isolated and migrated one at a time, following the Strangler Fig pattern [12] in the operational form described by Microsoft [13], where a routing facade directs each request to either the legacy or the modernized implementation depending on what has been migrated so far.

The pattern is well established and this paper claims no novelty in it. What the pattern does not address, and what proved to be the difficulty, is that the thing being strangled here contains authored content that keeps changing during the strangling.

4.3 Traffic routing and failback

Load-balancer switching moved traffic between tracks at slice granularity, with automated failback configured before each switch. Validation ran in lower environments first; production traffic moved only after the slice had been exercised somewhere it could not hurt anyone. This situates within the release engineering practices systematically reviewed by Shahin et al. [8], and the mechanics are unremarkable. The gate on the switch is the part that is not.

4.4 Content parity as the gate

Custom PowerShell and SQL scripts continuously synchronised content between tracks and maintained database backups, so that a rollback would not lose authored work. The requirement was zero content drift and zero data loss on rollback, which is a stronger guarantee than code migration usually needs, because a lost content edit cannot be recovered from a repository.

Listing 1 shows the classification at the centre of the parity check.

Listing 1. Drift classification between source and target content inventories. Workflow divergence is checked before revision or field divergence, because an item whose approval state has regressed is a records failure rather than a content bug.

```
    function Get-DriftClass {
  param($SourceItem, $TargetItem, [string[]]$GovernedStates)

  if ($null -eq $TargetItem) { return 'Missing' }
  if ($null -eq $SourceItem) { return 'Orphaned' }

  if ($GovernedStates -contains $SourceItem.workflow_state -and
      $SourceItem.workflow_state -ne $TargetItem.workflow_state) {
      return 'WorkflowDrift'
  }
  if ([int]$TargetItem.revision -lt [int]$SourceItem.revision) { return 'Stale' }
  if ($SourceItem.field_hash -ne $TargetItem.field_hash) { return 'FieldDrift' }

  return 'InSync'
}
```

A switch was permitted only when Missing, Stale, FieldDrift and WorkflowDrift counts were all zero. Orphaned items, present in the target but not the source, were reported without blocking, since a target may legitimately hold items authored after the slice began.

Figure 2 shows the resulting sequence, including the failback path.

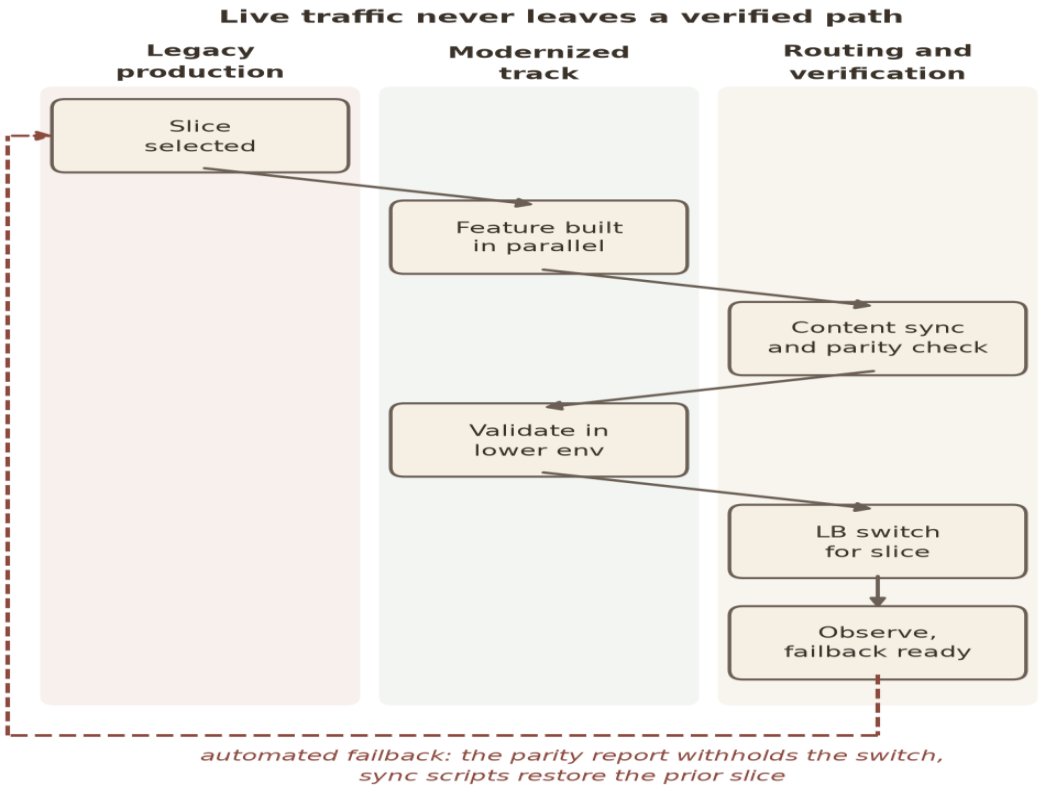

Figure 2. Dual-track cutover sequence. Each slice is built on the modernized track, synchronised and parity-checked, validated in a lower environment, then switched at the load balancer. The parity report withholds the switch when divergence is detected, and sync scripts restore the prior slice.

One detail of the ordering is worth calling out, because it was learned the hard way. Parity runs before validation rather than after it. Run the other way round, a slice with divergent content reaches the validation environment and gets signed off by a reviewer looking at content that is not what will actually ship.

5. Rebuild or Migrate: A Decision Framework

Not everything warranted rebuilding, and rebuilding by default is how modernization programmes exhaust their budget on subsystems that were working. Each subsystem was assessed on three axes: the feasibility of a headless rebuild, the capability gap between current and target state, and the delivery impact of leaving it unchanged. The operating rule was to rebuild where impact was high and the rebuild was tractable, and otherwise to migrate and refactor in place provided the underlying data schema was stable.

Listing 2 states the rule as applied.

Listing 2. Rebuild-versus-migrate decision rule and composite pressure score, weighting delivery impact double.

```
    $RebuildImpactFloor      = 4
$RebuildFeasibilityFloor = 3

function Get-Decision {
  param(
    [Parameter(Mandatory)][int]$ImpactScore,
    [Parameter(Mandatory)][int]$FeasibilityScore,
    [Parameter(Mandatory)][bool]$SchemaStable
```

```
    )

    if ($ImpactScore      -ge $RebuildImpactFloor -and
        $FeasibilityScore -ge $RebuildFeasibilityFloor) {
        return 'Rebuild headless'
    }
    if ($SchemaStable) { return 'Migrate and refactor' }
    return 'Rebuild headless'
}

function Get-RebuildPressure {
    <# Composite rebuild pressure, delivery impact weighted double. #>
    param(
        [Parameter(Mandatory)][int]$ImpactScore,
        [Parameter(Mandatory)][int]$GapScore,
        [Parameter(Mandatory)][int]$FeasibilityScore
    )

    $weighted = ($ImpactScore * 2) + $GapScore + $FeasibilityScore
    return [math]::Round($weighted / 4, 2)
}
```

Applying this across the estate produced Table 2.

Table 2. Rebuild-versus-migrate decision matrix. Scores are on a 1 to 5 scale, higher indicating greater pressure toward rebuild.

Subsystem	Feasibility	Gap	Impact	Pressure	Decision
Presentation layer	4	5	5	4.75	Rebuild headless
Search and indexing	4	4	5	4.50	Rebuild headless
Batch content operations	5	5	4	4.50	Rebuild headless
Analytics and journey tracking	3	4	4	3.75	Rebuild headless
Authentication and access control	2	2	3	2.50	Migrate and refactor
Editorial workflow and approvals	2	1	3	2.25	Migrate and refactor
Core information architecture	2	2	2	2.00	Migrate and refactor
Media and DAM assets	2	1	2	1.75	Migrate and refactor

Figure 3 shows the same assessment as a decision space

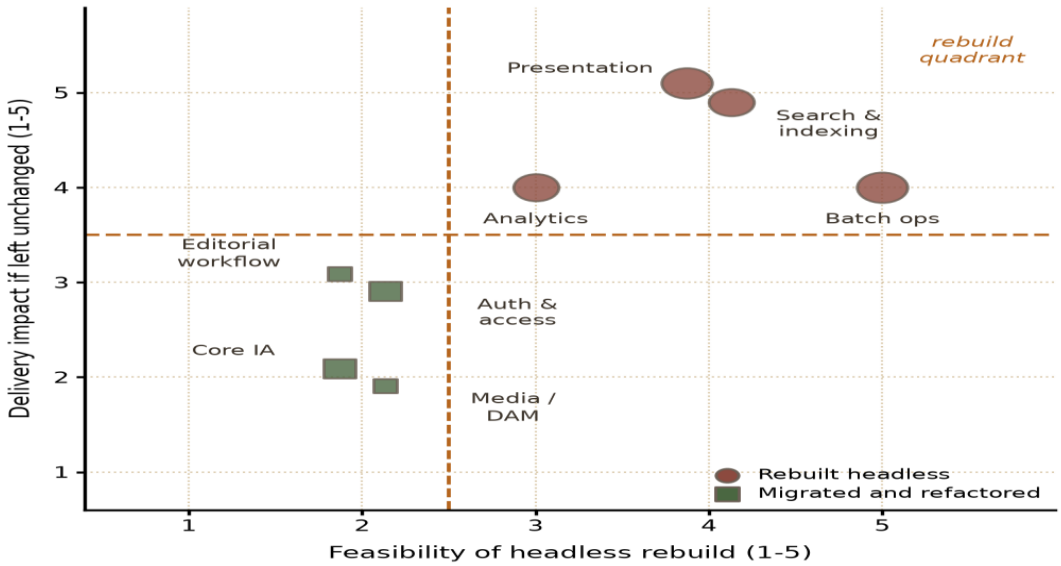

Figure 3. Rebuild-versus-migrate decision space. Marker area is proportional to capability gap. Points are offset where subsystems score identically on both axes.

The rebuilt set was the presentation layer, converted from legacy sublayouts into reusable React, Next.js and Vue.js components; search and analytics, moved onto a dedicated search service with custom caching, auto-updating index pipelines and personalised query support; and batch content operations, which had no bulk path at all. The migrated set was the core information architecture, whose templates, renderings and content items were extracted into a structured solution layout rather than rebuilt; media and digital asset management, connected through existing connectors because rebuilding risked asset reference integrity for no capability gain; and the governance and access subsystems discussed in Section 6.

The contrast with the formal decomposition criteria surveyed by Abgaz et al. [2] is worth naming. Those criteria are largely structural, concerned with coupling, cohesion and data ownership. The framework used here is explicitly consequentialist: it asks what breaks if this is left alone. For a content estate, where much of the coupling is between people and process rather than between modules, the consequentialist question produced better decisions than a structural one would have. The editorial workflow subsystem is the clearest case. It is tightly coupled, structurally unattractive, and was correctly left alone.

6. Preserving Governance Through the Change

This is the section that distinguishes a regulated-sector migration from any other, and it is the part of the programme that had the least prior art to draw on.

In most modernization accounts, the invariants are functional correctness and acceptable performance. Here the invariant set also included the approval regime itself. Content in this sector moves through defined states before publication, permission to publish is separated from permission to author, and the record of who moved what to which state and when is subject to inspection. SEC Rule 17a-4 [14] sets out the shape of the obligation for electronic records: preservation of the original record, a maintained index, and availability for regulatory production. Comparable obligations attach elsewhere in the sector.

The design consequence is that these could not be reconstructed after the migration. They had to survive it.

Four mechanisms carried the regime across.

Workflow states preserved in the authoring environment. Draft, Review and Approved states were maintained in the authoring environment before publication to delivery environments, unchanged in semantics across the platform

change. This is why the parity check in Section 4 tests workflow divergence before it tests content divergence: an item that arrives in the target with its approval state regressed has broken the record even if every field matches.

Approval gates on code as well as content. Check-in governance required mandatory code review and approval before changes were committed. The symmetry is deliberate. A platform whose content is governed but whose code is not has moved the control weakness rather than removed it.

Restricted publishing permissions and audit logging. Publishing rights were held separately from authoring rights, transitions were logged, and custom debugging layers were exposed only to authenticated authors rather than being generally available.

Attribute-based access control. Enterprise Windows Authentication and single sign-on integration restricted access according to internal department and location attributes, so that authorisation followed organisational structure rather than being separately administered.

Figure 4 shows the resulting pipeline.

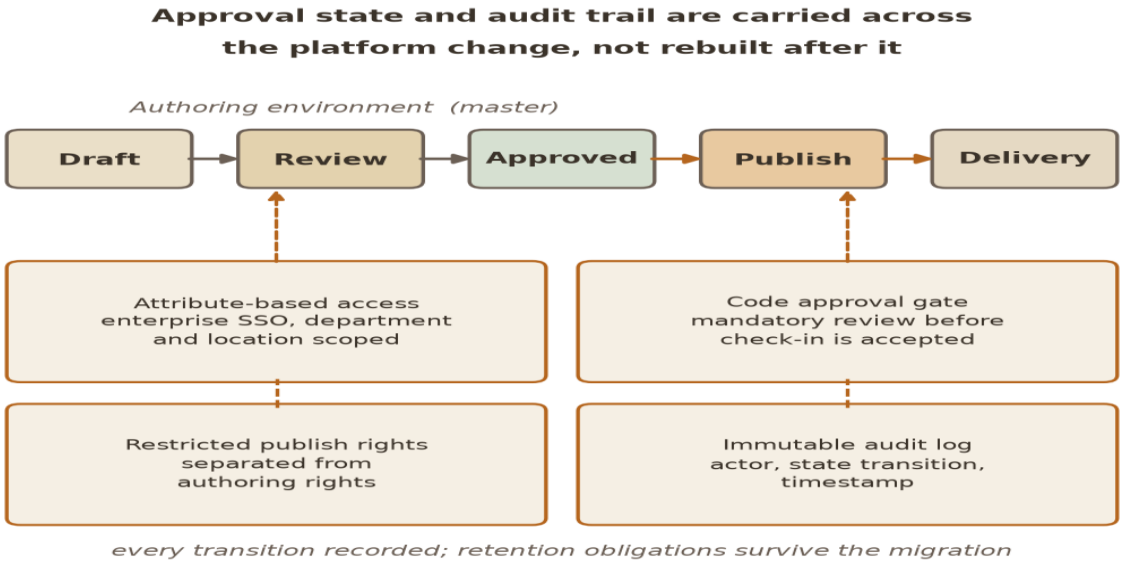

Figure 4. Governance-preserving publishing pipeline. Approval state transitions occur in the authoring environment; publication to delivery is a separately permissioned action; every transition is recorded.

The general lesson is that in a regulated estate the governance model belongs in the migration's definition of done, rather than on the list of things to restore once the platform is up. If a slice ships correct content but the transitions that produced it were not recorded, the migration has left a gap that is considerably more awkward to explain to an auditor than a missing page would have been.

7. What Took Longer Than Expected

Three things consistently exceeded their estimates, and they are offered as the paper's practical payload.

Content and taxonomy cleanup. Unpacking years of unstructured content and legacy URL dependencies always takes longer than planned. Content that has accumulated without a maintained taxonomy has no reliable inventory, and building one is discovery work with an unknown endpoint rather than migration work with a countable backlog. This item is first because it was the largest and the most consistently underestimated.

Third-party integration alignment. Coordinating data schemas across external services covering authentication, point of sale, payment and customer relationship management required deeper impact analysis than anticipated. Each

integration encodes assumptions about the platform it was built against, and those assumptions surface only when the platform changes.

API performance analysis and redesign. Where custom APIs are involved, performance analysis is not optional. The specific finding, and the one most transferable: prefer bulk fetching over individual item calls wherever a batch or list retrieval is possible. N+1 query patterns and per-item endpoint calls accumulate latency quickly under load, and they do so invisibly during development, where the collection has ten members rather than ten thousand. This behaviour is catalogued among service anti-patterns by Taibi and Lenarduzzi [5], and encountering it here in a content delivery context suggests the taxonomy transfers beyond the service architectures it was derived from.

The corresponding recommendations are to invest up front in a disciplined architectural convention for the solution structure, to establish front-end driven setups and starter kits early rather than after developers have absorbed the friction, and to write automated migration and backup scripting before it is needed rather than during an incident. Bogner et al. [4] find that post-migration maintainability depends more on sustained architectural convention than on the migration event, which matches what this programme observed.

8. LIMITATIONS

The limitations are substantial and are stated without hedging.

Single organisation, no control. This is one estate, one team, one sector. There is no comparison group, and no way to separate the effect of the architecture from the effect of the people who implemented it. Nothing here supports a general causal claim.

Reported rather than measured outcomes. No controlled measurement protocol was run and no baseline instrumentation was retained. What the programme observed is reported here rather than demonstrated against a comparison condition, and nothing in this account should be read as a measured result.

Sector and vendor specificity. The account is grounded in one content management ecosystem and one regulatory context. The sequencing strategy and the governance-invariant framing should transfer. The specific tooling will not.

9. CONCLUSION AND FUTURE WORK

A content platform in a regulated sector can be modernized without interrupting service. The part that made it work, though, was not the migration pattern. Strangler Fig sequencing is well understood and available to anyone, and on its own it would not have been enough here. What carried the programme was the gate on each switch: a content parity check that refused the cutover until source and target agreed on every governed dimension, approval state included.

Two findings seem worth carrying elsewhere. Rebuild decisions came out better when they were made consequentially, by asking what actually breaks if a given subsystem is left alone, than they would have from a structural assessment, because a great deal of the coupling in a content estate runs through people and process rather than through modules. And governance belongs in the definition of done rather than on a post-migration cleanup list.

Assunção et al. [11] argue that the abundance of modernization strategy is itself a problem for practitioners seeking situated guidance. This paper is offered in that spirit: not as a general method, but as one account, with its limits and its gaps named.

Future work should address what this study could not. A multi-organisation study of content platform modernization would establish whether the decision framework in Section 5 generalises. Most valuable would be work on governance-preserving migration as a problem in its own right, drawing on records management scholarship on authenticity and chain of custody, which the software engineering literature has not yet met.

Appendix A: Accompanying Artifacts

The analysis and figures in this paper are reproducible from the artifacts accompanying it.

Artifact	Purpose	Feeds
content_parity_audit.ps1	Drift classification between source and target content inventories	Listing 1, Section 4
rebuild_or_migrate.ps1	Feasibility, gap and impact scoring across the estate; applies the decision rule and emits the matrix	Listing 2, Table 2, Figure 3
source_items.csv, target_items.csv	Worked example inventories for the parity audit	Listing 1
parity_report.csv	Classified drift output of the parity audit	Section 4
rebuild_migrate_matrix.csv	Scored decision matrix, output of rebuild_or_migrate.ps1	Table 2, Figure 3

Both scripts are PowerShell, which is the native automation surface for a .NET content platform estate of this kind, and they run unmodified on Windows PowerShell 5.1 and PowerShell 7. Neither requires an external module. Figure 3 is plotted directly from rebuild_migrate_matrix.csv, so the figure and Table 2 cannot diverge from the scoring script that produced them.

REFERENCES

1. M. Martínez Saucedo, G. Rodríguez, T. Gomes Rocha and R. Santos, "Migration of monolithic systems to microservices: A systematic mapping study," Information and Software Technology, 2025. doi: 10.1016/j.infsof.2024.107590
2. Y. Abgaz, A. McCarren, P. Elger, D. Solan, N. Lapuz, M. Bivol, G. Jackson, M. Yilmaz, J. Buckley and P. Clarke, "Decomposition of monolith applications into microservices architectures: A systematic review," IEEE Transactions on Software Engineering, 2023. doi: 10.1109/TSE.2023.3287297
3. P. Di Francesco, P. Lago and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in Proc. IEEE International Conference on Software Architecture (ICSA), 2018. doi: 10.1109/ICSA.2018.00012
4. J. Bogner, J. Fritzsch, S. Wagner and A. Zimmermann, "Industry practices and challenges for the evolvability assurance of microservices," Empirical Software Engineering, vol. 26, 2021. doi: 10.1007/s10664-021-09999-9
5. D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," IEEE Software, vol. 35, no. 3, 2018. doi: 10.1109/MS.2018.2141031
6. A. Balalaie, A. Heydarnoori and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," IEEE Software, vol. 33, no. 3, 2016. doi: 10.1109/MS.2016.64
7. S. Peltonen, L. Mezzalira and D. Taibi, "Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review," Information and Software Technology, vol. 136, 2021. doi: 10.1016/j.infsof.2021.106571
8. M. Shahin, M. Ali Babar and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," IEEE Access, vol. 5, 2017. doi: 10.1109/ACCESS.2017.2685629
9. N. Forsgren, M.-A. Storey, C. Maddila, T. Zimmermann, B. Houck and J. Butler, "The SPACE of developer productivity," ACM Queue, vol. 19, no. 1, 2021. doi: 10.1145/3454122.3454124
10. A. Noda, M.-A. Storey, N. Forsgren and M. Greiler, "DevEx: What actually drives productivity," ACM Queue, vol. 21, no. 2, 2023. doi: 10.1145/3595878
11. W. K. G. Assunção, L. Marchezan, A. Egyed and R. Ramler, "Contemporary software modernization: Perspectives and challenges to deal with legacy systems," in Proc. Software Engineering in 2030 Workshop (SE2030), 32nd ACM International Conference on the Foundations of Software Engineering, 2024. arXiv:2407.04017
12. M. Fowler, "StranglerFigApplication," 2004. [Online]. Available: https://martinfowler.com/bliki/StranglerFigApplication.html
13. Microsoft, "Strangler Fig pattern," Azure Architecture Center, 2024. [Online]. Available: https://learn.microsoft.com/azure/architecture/patterns/strangler-fig
14. U.S. Securities and Exchange Commission, "Rule 17a-4: Records to be preserved by certain exchange members, brokers and dealers," 17 CFR 240.17a-4.