

FINANCIAL SERVICES

The platform model paradox:

Why operating model transformations don't
accelerate delivery over legacy systems



Executive summary

A fundamental contradiction in enterprise transformation: implementing platform or product operating models over legacy IT systems rarely delivers promised velocity improvements. When cross-functional product teams and modern governance sit atop unchanged legacy architectures with monolithic databases and batch-processing constraints, nothing moves faster.

The core issue stems from a structural mismatch articulated by Conway's Law: organisations restructure teams around modern product thinking while underlying technical systems remain architected for an earlier era.

This creates a 'chicken-and-egg problem' teams need modern infrastructure to operate at modern velocity, but legacy systems constrain the very modernisation that would enable speed.

Research indicates that 70% of operating-model transformations fail to meet objectives, with technology-organisation misalignment cited as the primary cause [1]. When enterprises implement product operating models without addressing legacy constraints, they experience 'innovation drag'. The gap between organisational aspiration and technical capability widens, and velocity paradoxically decreases as coordination overhead compounds existing bottlenecks.

This paper looks to explain why this paradox occurs, quantifies its impact and outlines strategies that acknowledge the inherent constraints that legacy systems impose on modern operating models.

The structural problem: operating model vs. technical reality

Conway's Law and the architecture-organisation mismatch

Conway's Law observes that organisations which design systems produce designs mirroring their communication structures [2]. The inverse holds equally: when organisational structures change but technical architectures remain static, tension emerges throughout the system [3].

Product operating models reorganise teams around customer-facing capabilities with autonomous cross-functional units. Teams expect to deploy changes independently on rapid cadences measured in days or weeks [4]. However, legacy systems were architected for functional silos, centralised IT control and release cycles measured in months or quarters [5][6].

A financial services firm reorganised into product squads responsible for mortgage origination, but the underlying mainframe processes loan applications through overnight batch jobs with quarterly release windows. The squad's velocity is fundamentally constrained regardless of their Agile ceremonies or team structure [1]. This creates "organisational debt" – the structural misalignment between how teams should work and how underlying systems force them to work [6].



The platform model speed illusion

The platform operating model promises speed through separation of concerns: stable-platform teams provide reusable capabilities while stream-aligned teams rapidly compose solutions from these building blocks [8][9]. In theory, this decoupling enables innovation at the edge while maintaining core stability.

This two-speed approach works only when the 'slow' layer can support the 'fast' layer's needs. When legacy platforms require specialist intervention for minor changes, when integration requires custom connectors or when data access demands months-long governance processes, the platform layer becomes a bottleneck rather than an enabler [10].

A European bank implemented cross-functional product teams but maintained centralised release pipelines managed through legacy systems. The result: long delays in getting features to production – the benefits of organisational agility were negated by technical constraints. The bank only achieved velocity improvements years later after pursuing technological transformation alongside organisational change [11].

The measurement gap

Organisations measure operating model success through lagging indicators – customer satisfaction, revenue growth – while ignoring the leading indicator that legacy systems provide technical cycle time. When simple features take weeks due to crossing multiple legacy systems, the problem is architectural, not organisational.

One financial services provider tracked precisely: their legacy monolithic app had a median cycle time of 20 days, while new micro-apps built with modern architecture achieved 9–10 day cycles – a 50% improvement in delivery speed simply from architectural changes, with identical team structure and processes [12]. This quantifies what executives sense but cannot measure: legacy foundations constrain velocity regardless of organisational design.

Why legacy systems negate operating model benefits

The monolithic architecture problem

Legacy systems typically feature tightly coupled components where changes ripple across the entire codebase [10]. This creates several velocity constraints that product operating models cannot overcome:

Deployment rigidity: Even minor bug fixes require redeployment of the entire application, increasing risk and discouraging frequent releases [10]. A product team may complete a feature in days, but monthly deployment windows mean the feature sits idle, negating agile velocity.

Integration bottlenecks: Modern tools rely on APIs and microservices, but legacy platforms use proprietary data formats and lack integration capabilities [13]. Building custom connectors becomes a hidden tax that every product team pays, with analysis noting these integrations are “costly and brittle, leading to maintenance overload” [10].

Testing burden: Monolithic systems require extensive regression testing because any change could impact the entire application [10]. Product teams experience long test cycles where testing becomes the bottleneck because the underlying architecture lacks modularity.

Scalability constraints: Legacy architectures cannot scale specific components. Under load the entire application must scale, which is “difficult and expensive” [10]. When a feature succeeds and drives traffic, the legacy system becomes the limiting factor.

The batch processing trap

Many legacy systems, particularly mainframes that remain “the backbone of many large enterprises,” operate on batch-processing models [13]. Transactions accumulate during the day and process overnight, fundamentally incompatible with continuous deployment expectations [11].

This creates ‘release cycle incompatibility’: Agile teams work in two-week sprints, but legacy batch systems deploy quarterly. The Agile team can complete work faster, but deployment velocity remains constrained by the underlying platform’s release cadence. As one practitioner observed: “What’s the use of Agile if you’re waiting for a waterfall-driven business requirements team?” [14]

The mainframe development community has attempted to address this through release-based processes adapted for Agile teams. However, these approaches acknowledge the fundamental constraint: mainframe systems can adopt some Agile practices, but batch processing dependencies and longer release cycles remain inherent to the architecture [13].

The data architecture bottleneck

Operating model transformations frequently overlook data architecture, leaving “data in the hands of old functional constructs” [1]. One consumer goods company established cross-functional product teams but maintained a centralised data department that “did nothing to democratise access,” creating bottlenecks that “reduced the efficiency of Agile teams” [1].

Legacy data architectures present several constraints: information trapped in departmental databases requires complex ETL processes; fixed database structures resist adding new fields or relationships; and batch updates create synchronisation issues where product teams building real-time features receive stale information [13].

The dependency problem

Legacy applications accumulated over decades contain complex, often undocumented integrations [15]. When organisations restructure into product teams aligned with customer journeys, they discover that these journeys cut across multiple legacy systems with intricate dependencies [8].

A banking transformation provides illustration: the organisation restructured teams around customer capabilities, but these capabilities required changes across 5–7 legacy systems, each owned by different parts of the organisation [6]. The result: “legacy apps will fight your transformation and cause software teams to manage dependencies between teams, wait for changes to be made and slow down progress” [6].

This creates coordination overhead that scales non-linearly with teams and dependencies [16]. Organisations implement Agile to reduce coordination costs, but legacy system dependencies increase coordination requirements – the operating model transformation produces the opposite of its intended effect.

The pace-layered strategy and its limitations

Gartner proposed the “Pace-Layered Application Strategy,” categorising systems into three layers based on change frequency [17]:

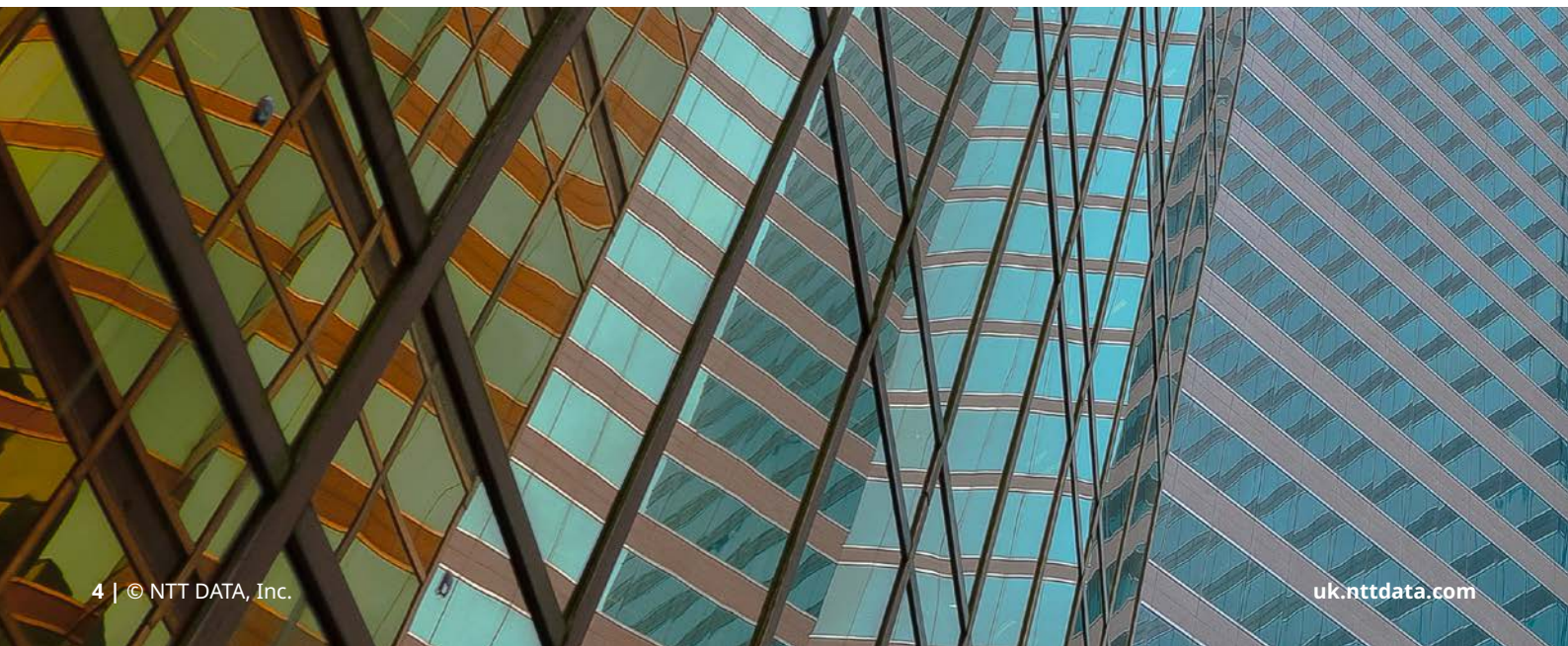
- **Systems of record** (change every 5–10 years): Core transaction platforms, ERP, mainframes.
- **Systems of differentiation** (change every 1–3 years): Business capabilities providing competitive advantage.
- **Systems of innovation** (change every 2–3 months): Experimental solutions for emerging requirements.

This framework acknowledges that not all systems should operate at the same velocity. However, ThoughtWorks Technology Radar assessed pace layering and concluded it creates an “unhelpful focus on layers within an architecture” because “many types of change cut across multiple layers” [18].

A product team wants to add a new product line to an e-commerce platform. In pace-layered thinking, this appears quick and experimental. In reality, it requires updating the product catalogue database (system of record), modifying inventory management integration (system of differentiation), changing warehouse systems for new SKUs (system of record) and updating the CMS for product presentation (system of innovation). The velocity of this change is constrained by the slowest layer it must touch – typically the system of record with quarterly release cycles [8].

Furthermore, pace layering tends to calcify legacy system constraints by labelling them as intentionally slow [1]. Systems designated as ‘stable cores’ remain untouched, with organisations investing in fast-changing wrappers around slow-changing foundations. This creates the illusion of velocity while perpetuating the underlying problem.

Bimodal IT, a related concept suggesting separate ‘modes’ for stability and speed, faces similar critiques. Martin Fowler concludes it is a “flawed notion” because software cannot be neatly divided into two categories; most systems require both stability and agility. In practice, bimodal approaches often devolve into ‘multi-speed IT’ where different parts of the organisation operate at different velocities, creating friction and handoffs [19].



The decoupling strategy paradox

The canonical solution to legacy constraints involves introducing abstraction layers – API gateways, microservices and integration platforms – that decouple modern product teams from legacy backends.

These strategies offer genuine benefits: centralised security and authentication without modifying legacy code; protocol translation making legacy systems accessible to modern clients; rate limiting to protect legacy systems from overload and caching to reduce load and improve perceived performance [20].

However, decoupling strategies face fundamental limitations that prevent them from fully eliminating legacy constraints:

The last-mile problem: API gateways can make legacy systems accessible, but they cannot make them fast.

If the underlying mainframe takes 45 seconds to process a transaction, the API gateway adds a modern interface to a slow system – the response time remains 45 seconds plus gateway latency. Users experience slower performance through a modern API than through the legacy interface.

Limited functional change: Integration layers enable calling existing legacy functions, but they cannot add capabilities that the legacy system lacks. If the legacy system processes refunds through overnight batch jobs, an API gateway cannot make it real time – it can only provide modern syntax for queuing the batch job.

Complexity accumulation: Each abstraction layer adds operational complexity, monitoring requirements and failure modes. Organisations discover that they now maintain both legacy systems and a complex integration infrastructure, increasing rather than reducing technical debt [11].

The ‘strangler fig pattern’ – gradually replacing legacy components by building new functionality alongside old – shows promise with some teams reporting 50% cycle time improvements. However, full strangler implementations take 18–24 months to show maximum value, and many organisations “pause new feature development until core services are fully rewritten” [21] – exactly the velocity reduction product operating models seek to avoid.

After platform modernisation with API-led decoupling, organisations achieve “2x faster feature delivery after eliminating bottlenecks” and “92% improvement in deployment speed” [22]. These are significant gains, but they require actually modernising the systems, not merely wrapping them in APIs. The improvement comes from architectural changes, not organisational restructuring.



Quantifying the operating model-legacy gap

A critical insight emerges from McKinsey's research: transformations succeed when "companies improve their capabilities in automating operations and digitising business processes" and technology changes accompany organisational changes.

The most revealing finding: organisations with strong digital cultures still face a 40% failure rate, often because they "neglect to conduct thorough analysis of their business environment or establish clear vision and strategy" [24]. Translation: even digitally mature organisations fail when they don't align operating model changes with underlying technical capabilities.

Transformation failure rates

Research documents alarmingly high failure rates for operating model and digital transformations:

70%

of digital transformation initiatives fail to meet objectives [23]

88%

of business transformations fail to achieve their original ambitions [23]

63%

of operating model transformations achieve "most objectives" but only 24% are highly successful [1]

Velocity impact measurements

The pattern is consistent: operating model changes alone produce modest or negative velocity impacts, while operating model changes combined with architectural modernisation produce dramatic improvements. The architecture is the enabler; the operating model is the amplifier.

Organisations that measure cycle time before and after operating model changes reveal the legacy constraint quantitatively:

Before addressing legacy systems:

- Legacy monolithic applications: 20-day median cycle time [12]
- Simple features taking weeks due to team dependencies, not complexity [6]
- Development teams spend more time maintaining than building [16]
- 25% of technical budget diverted to resolving legacy-related issues [25]

After architectural modernisation:

- Micro-apps with modern architecture: 9–10-day cycle time (50% improvement) [12]
- Platform optimisation: 2x faster feature delivery, 25–40% cloud cost reduction [22]
- Organisations managing technical debt: 50% faster delivery times [25]
- IT operating model shifts combined with architecture: 200% boost in software development productivity [26]

Why organisations persist in the paradox

The organisational change illusion

Operating-model transformations appeal to executives because they appear controllable, measurable, and relatively fast [1]. Reorganising teams, redefining governance, and implementing new processes can occur in months. By contrast, modernising a mainframe that processes millions of daily transactions feels risky, expensive, and indefinite in duration [27].

This creates a preference for tangible action over uncertain investment. Executives implement product-operating models because they can demonstrate new team structures, agile ceremonies, product roadmaps. The fact that underlying systems remain unchanged gets rationalised: “We’ll address technical debt later” or “Our focus is on customer experience, not infrastructure” [25].

Organisations fall into what one analysis calls the “vicious circle” of legacy systems: “the complex, monolithic nature not only makes it difficult to make changes, but introduces further complexity. This leads to high levels of technical debt, which makes maintenance more difficult and expensive” [11]. Each workaround makes the next change harder, yet organisations continue investing in operating-model transformations rather than addressing the root architectural constraint.

Vendor and consultant incentives

The transformation consulting industry has limited incentive to highlight the operating-model-legacy gap. Operating-model engagements are lucrative, politically palatable, and replicable across clients. By contrast, recommending legacy modernisation requires specialised technical expertise and extends engagement timelines.

One practitioner observes that “the idea that every technology implementation has to be a multi-year project staffed by external consultants is a holdover from another era” [27] – yet consultants continue proposing multi-year operating-model transformations while treating legacy systems as unchangeable constraints.

The sunk cost trap

Legacy systems represent enormous sunk investments – millions or billions of dollars over decades [13]. Executives face a psychological barrier to admitting these systems must be fundamentally changed or replaced [27]. The prevalent attitude: “if it works, why change it” [28].

This ignores the fact that “legacy system” describes systems that constrain the business when it needs to change. A “poorly architected system can be regarded as legacy on its day of launch” if it cannot support required evolution [28]. The question isn’t whether legacy systems work – they often do – but whether they enable the velocity that modern operating-models require.

Strategies that acknowledge constraint

Honest velocity road mapping

Organisations that successfully navigate the operating-model-legacy gap start with brutal honesty about technical constraints [6]. This means:

Mapping dependencies before restructuring:

Using techniques like domain-driven design context mapping to understand how capabilities actually flow through systems, revealing where legacy dependencies will constrain velocity before teams experience bottlenecks [29].

Measuring actual cycle times: Tracking how long features truly take from concept to production, including waiting time for legacy system changes [12]. This makes the cost of legacy constraints visible in business terms.

Setting realistic velocity expectations: Being transparent that product teams will achieve modest velocity improvements until architectural constraints are addressed [30]. This prevents demoralisation when teams adopt agile practices but see no improvement.

One organisation that took this approach reported:

“Development as Agile but run the rest of the product cycle in a traditional sense if it works better for your company” [14]. This pragmatic stance acknowledges that Agile teams can work internally at high velocity while accepting that external release cadence remains constrained by legacy systems.

Parallel architectural investment

Organisations that succeed with operating-model transformations pursue simultaneous architectural modernisation, not sequential “org first, tech later” approaches [1]. Research is unambiguous: “Only after the bank pursued a technological transformation years later did it get the benefits of speed and stability it had hoped to unleash from the start” [1].

This parallel approach involves:

Iterative decoupling: Using strangler fig patterns to gradually replace high-friction legacy components while maintaining business continuity [21]. Organisations target specific bottlenecks that constrain velocity, for example, moving real-time inventory from overnight batch to event-driven architecture – rather than attempting wholesale replacement.

Platform team focuses on legacy integration:

Structuring platform teams to actively manage and progressively simplify legacy integration, not just build new capabilities [31]. This acknowledges that “platform modernisation” means making legacy systems behave more modernly.

Architectural runway investment: Continuously investing in foundational capabilities that enable future velocity [9]. This treats architecture as a product backlog item with business value measured in cycle time reduction.

One financial services provider achieved this balance: “shifting its IT operating-model from system-based silos to cross-functional teams while modernising technology stack” produced 200% productivity improvement in three months [26]. The combination of organisational and architectural change produced results neither would achieve alone.

Value stream-aligned modernisation

Rather than pace layering by system type, leading organisations prioritise modernisation by value stream velocity impact [32]. This means:

Identifying bottleneck systems: Measuring which legacy systems constrain the most valuable customer journeys [6]. If mortgage origination generates 40% of revenue but depends on a mainframe with quarterly releases, that system gets modernisation priority.

Progressive modernisation of hot paths: Refactoring the specific code paths and data flows that high-value features traverse most frequently [20]. This provides velocity improvement where it matters most – in active product development.

Modernisation as product work: Treating legacy modernisation as product features that enable other features, not separate IT projects [30]. Product teams include modernisation stories in their backlogs because they directly impact delivery velocity.

Organisations following this approach report “reduced lead time for changes from 2 months to 1-day, improved failure rate from over 30% to below 10% and saved 50% of cloud costs” [21].

Conclusion: naming the elephant

The uncomfortable truth is this: implementing platform or product operating models over legacy IT systems is organisational theatre where velocity improvement is the goal. The reorganisation provides the appearance of transformation while the technical architecture remains the constraint on actual delivery speed.

This is not an argument against modern operating models – product thinking, cross-functional teams and customer-centric organisation provide genuine value in alignment, decision making, and strategic clarity. However, these organisational benefits do not include velocity improvement when legacy systems constrain deployment frequency, require specialist intervention for changes, process through batch cycles or create complex inter-system dependencies.

The data is unambiguous: 70%+ of transformations focusing on organisation without technology fail to meet objectives. Organisations that modernise architecture achieve 2x velocity while those that only reorganise see marginal or negative impacts. Technical constraints consume 25–40% of team capacity regardless of organisational structure. Legacy-constrained teams take 20 days for changes that modern architectures complete in 9 days.



Three paths forward exist for organisations confronting this paradox:

1. Honest expectation setting: Implement product operating models for their organisational benefits while being transparent that velocity improvements will be modest until architectural constraints are addressed. This prevents transformation fatigue when promised speed doesn't materialise.

2. Parallel transformation: Pursue organisational and architectural modernisation simultaneously, accepting that this requires more upfront investment but delivers actual velocity improvement. This is the approach successful transformations take.

3. Selective application: Apply product operating models only to domains where underlying systems can support independent deployment and rapid change. Maintain different operating approaches for domains constrained by legacy systems, avoiding friction from imposing product governance on teams who cannot achieve product velocity.

Velocity comes from architecture more than organisation. Conway's Law tells us that systems mirror organisational structure – but the inverse is equally true: organisational velocity cannot exceed what the system architecture permits. A cross-functional product team with daily standups will deliver features at the pace the underlying mainframe allows, not at the pace Agile methodology promises.

The elephant in the room is that most organisations choose organisational transformation because it's easier to execute than architectural modernisation. Until enterprises acknowledge that the primary constraint is technical, not organisational, they will continue experiencing the paradox: nothing goes faster because the same underlying slow-release cycles exist.

Transformation success requires naming this elephant explicitly, measuring its impact quantitatively and investing in the architectural changes that actually enable velocity – not as a future phase, but as the foundation without which operating model transformation delivers motion without meaningful progress.

Authors and contributors



Matt Kearney

Matt is Head of Insurance Pensions & Investments Consulting NTT DATA UK&I. He has 25 years of experience in strategy development and organisational transformation across a range of sectors, including telecoms and logistics. He is also a regular speaker on business agility.



Mark Toye

Mark is the CTO for NTT Data UK&I Insurance unit, with over 20 years' insurance consulting experience and a proven track record of designing and delivering strategy and change with complex data processing, AI and integration solutions.



Ravi Purushothaman

Ravi leads the UK&I Enterprise Agility Practice, bringing over 25 years of experience delivering complex transformations, embedding cultures of continuous improvement and a deep expertise in platform and product delivery.

Bibliography

- [1] **McKinsey & Co (2025)** 'How to get your operating model transformation back on track'
- [2] **Fowler, M. (2022)** 'Conway's Law', Martinfowler.com.
- [3] **Learning Loop (2023)** 'Architecture and Organisation alignment'.
- [4] **Thinslices (2025)** 'Autonomous Product Teams'.
- [5] **IBM (undated)** 'Mainframe Agile Development'.
- [6] **Liminalarc (2022)** 'Legacy Systems and Organisational Debt'.
- [7] **Iteo (2024)** 'Domain-Driven Design and Organisational Alignment'.
- [8] **Thoughtworks (2023)** 'Platform Operating Models and Legacy Systems'.
- [9] **Scaled Agile (2026)** 'SAFe Platform Team Patterns'.
- [10] **Vfunction (2024)** 'Monolithic Architecture Constraints'.
- [11] **Myshyft (2025)** 'Monolithic System Constraints'.
- [12] **Swimm (2025)** 'Cycle Time and Architectural Impact'.
- [13] **Confluent (2014)** 'Legacy Architecture and Integration'.
- [14] **Muller (2018)** 'Agile Release Cycles and Legacy Systems'.
- [15] **Nextpathway (2026)** 'System Dependencies in Legacy Environments'.
- [16] **Oko One (2025)** 'Technical Debt and Maintenance Burden'.
- [17] **Leanix (2023)** 'Pace-Layered Architecture', Leanix.
- [18] **Thoughtworks (2015)** 'Pace-Layered Application Strategy Critique'.
- [19] **Devops (2022)** 'Bimodal IT and Multi-Speed Architecture'.
- [20] **LinkedIn (2025)** 'API Gateway Benefits and Limitations'.
- [21] **Future Processing (2025)** 'Strangler Fig Pattern and Modernisation'.
- [22] **Sphere Inc. (2025)** 'Platform Modernisation Impact'.
- [23] **Mendix (2025)** 'Digital Transformation Failure Rates'.
- [24] **Altix Consulting (2024)** 'Digital Culture and Transformation'.
- [25] **McKinsey & Co (2023)** 'Technical Debt and Organisational Impact'.
- [26] **Skyone (2025)** 'Operating Model and Architecture Combined Transformation'.
- [27] **Ahmedsobhan (2025)** 'Legacy Modernisation Barriers'.
- [28] **Mozaic (2025)** 'Legacy Systems Definition'.
- [29] **Iteo (2024)** 'Domain-Driven Design Techniques'.
- [30] **Savvycoms (2025)** 'Realistic Agile Velocity Planning'.
- [31] **Mia Platform (2025)** 'Platform Team Modernisation Focus'.
- [32] **Moderndata101 (2025)** 'Value Stream Architecture Alignment'.

