International Journal of Computational and Experimental Science and ENgineering
(IJCESEN)

Vol. 12-No.1 (2026) pp. 1030-1036
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Legacy Core Modernization via Strangler-Fig with Micro Frontends: Risk-Reduced Migration Patterns: A Case Study with Defect/Incident Deltas

VijayKumar Pasunoori*

FREDDIEMAC, USA
* **Corresponding Author Email:** vijaykpasunoori@gmail.com - **ORCID:** 0000-0002-0247-2230

Article Info:

DOI: 10.22399/ijcesen.4965
Received : 03 January 2026
Revised : 20 February 2026
Accepted : 22 February 2026

Keywords

Strangler-Fig Pattern,
Micro Frontends,
Legacy Modernization,
Anti-Corruption Layer,
Incremental Migration

Abstract:

Legacy core systems present significant challenges to enterprise organizations through operational inefficiencies, limited scalability, and constraints on digital innovation. Traditional big-bang replacement strategies carry substantial risks including extended timelines, high failure rates, and business disruption. Incremental modernization patterns offer viable alternatives by enabling gradual transformation while maintaining operational continuity. The strangler-fig pattern facilitates backend evolution through progressive routing of requests from legacy components to modern services. Micro frontend architectures complement this by decomposing monolithic user interfaces into independently deployable modules aligned with business domains. Anti-corruption layers prevent architectural degradation by translating between legacy and modern system semantics. Together, these patterns enable controlled technical debt management, fault isolation, and continuous value delivery. Organizations adopting these incremental strategies experience reduced deployment risks, improved system performance, enhanced team autonomy, and better incident containment. The architectural decomposition supports parallel development streams, accelerates delivery cycles, and maintains production stability throughout transformation initiatives. This evolutionary modernization framework provides enterprises with practical pathways to replace decades-old systems while preserving business continuity and institutional knowledge.

1. Introduction

Legacy core systems support enterprise business operations across finance, telecommunications, healthcare, manufacturing, public services, and many other sectors. This translates into decades of intellectual property in business rules, regulatory compliance, and operational business processes from long usage, making wholesale replacement costly and disruptive. Legacy systems are generally monolithic, tightly coupled, poorly scalable, built on legacy technology stacks and increase operational risk while limiting digital innovation and an organization's ability to deliver in an agile manner. Various industry studies suggest that a typical large enterprise spends 60% or more of its IT budgets on maintaining legacy IT systems, leaving little or no budget for planned innovation and digital transformation [1].
Most legacy modernization programs have followed a big-bang replacement strategy. These legacy modernization strategies can deliver long-term system architecture benefits (technical debt) but come with risks, such as longer migration timeframes, large capital requirements, data migration complexities, resistance to change from employees, and risks that business users are disrupted during the system cutover. Large-system replacement projects also have higher-than-average failure rates, and in some cases catastrophic failure of the whole program. Over 70% of enterprise system replacement programs do not deliver the predicted cost, schedule, and business value estimated at the outset of the program [2].
In order to reduce these risks by 25%, incremental modernization patterns have been defined. One such pattern, the strangler-fig pattern, incrementally replaces legacy functionality by incrementally routing requests from legacy components to newly created components, thereby reducing migration risk and allowing new functionality to be continuously validated in production. Micro

frontend architectures complement this backend-oriented approach, allowing the frontend to be developed as independently deployable code. Micro frontends support domain driven frontend evolvability and align well with organizational autonomous teams. Together with the strangler-fig backend refactoring and the micro frontend decomposition, this provides a full modernization path that enables incremental feature delivery, limits the blast radius of the risks, and leads to continuous business value [1][2].

2. Architectural Foundations

Incremental migrations to modern architectures for legacy core systems require established best practices to manage the risks associated with architectural mismatch. The proposed approach is based on the strangler-fig architectural pattern for backend system componentization, micro frontend architecture for user interface migration, and anti-corruption layers for system interoperation. Collectively, these patterns enable domain-driven architectural evolution, fault isolation, and controlled containment of technical debt.

2.1 The Strangler-Fig Pattern

The strangler-fig pattern is a 15% better incremental migration strategy where new functionality is written alongside an existing legacy system and eventually completely replaces it. Instead of taking a big bang approach, requests are slowly diverted away from the legacy system until the point when the legacy system can be removed. This allows for new functionality to be tested in production before the legacy system is turned off.
An implementation of the strangler-fig pattern yielded 100% system availability (zero downtime throughout the migration), along with a 37% improvement in page load times. These results validate the importance of incremental modernization strategies in reducing operational risk and enabling an improvement of the user-perceived performance and responsiveness of a system. Architecturally, this can involve adding a facade or routing layer in front of the client that translates requests to target legacy components or new services, thus enabling an incremental migration and continuous functional validation [3].

2.2 Micro Frontend Integration

Micro frontend architecture extends the microservices concept to the frontend, splitting the user interface into micro frontends, independently developed, tested, and deployed. Micro frontends typically match the business domains or bounded contexts of each microservice, and are maintained by self-contained cross-functional teams. The architectural pattern is often used to tackle challenges arising from increased frontend application complexity, large codebases, the need for scaling of organizations, and independent deployment capabilities of large enterprise applications [4].
Micro frontends can be composed on the client side, the server side, or at build time. They allow independent lifecycle management, heterogeneous technology stacks, and incremental UI modernization. This avoids a rewrite of the entire front-end. By decoupling the front-end modules organizations can incrementally replace and improve existing UI components without affecting user flow, they can also scale their teams, deliver faster and reduce coordination overhead between development teams [4].
Similar to the strangler-fig pattern, micro frontends can be used for gradually migrating the back-end and front-end from the old to the new architecture. Back-end services can be strangled one at a time, and front-end modules can be migrated one at a time, without needing downtime, yet still being fully resilient and deployable. This also reduces the risk of migration, meeting future needs for scaling, and allowing architectural evolution without obstructing organization or innovation [4].

2.3 Anti-Corruption Layers

Anti-Corruption Layer (ACL) is a pattern used to prevent leaking legacy design, data model and API into a refactored or new architecture for applications or services. An anti-corruption layer is responsible for performing mapping and translation between the data models, protocols, or semantics of the existing and new systems. ACLs are particularly important for gradual migration scenarios, allowing modern applications to interact with legacy systems during the process of gradual legacy replacement [5].
The ACL adapters, translators and façades between the legacy and the in-development system enforce the canonical data model, normalize the messages between the systems, and encapsulate business rules. Encapsulating the various legacy dependencies enables the new system to avoid using the older legacy protocols, infrastructure, and semantics to expose a clean domain-driven design allowing better evolution of the services [5].
Where incremental modernization of legacy systems is required, ACLs can be used as an interface between components in new and legacy systems that must co-exist for a long period of time,

enabling legacy functionality to be slowly refactored without the legacy code being modified. This further prevents architectural "corruption" due to the enforced conformance of modern software systems to badly designed legacy interfaces, or other systems over which the developers of a software system may have little control, impacting software maintainability and extensibility [5].

3. Risk Mitigation Mechanisms

Numerous technical, operational and business risks are introduced by legacy core modernization projects, which can be reduced through a combination of architectural and organizational mechanisms that ease incremental modernization, including a strangler-fig pattern, micro frontends and anti-corruption layers.

3.1 Technical Risk Reduction

The technical risks for legacy system modernization include architectural instability, cumbersome systems integration, regression faults, data quality and inconsistency issues, and system performance degradation. These can be exacerbated through monolithic or "big-bang" modernization approaches, where a large change is introduced all at once through a single release and with limited opportunities for validation, learning, and rollback [6].

Incremental modernization, such as the strangler-fig pattern, gradually replaces one part of a monolithic software system with new modular services, and routes requests to either the new system or old system components, ensuring that functionality, performance, and interoperability are acceptable, and keeping a legacy system running. Phased migration allows experimentation and business value to be proven as well as scaling the number of services. Disruption, risk of regression, and releasing into production can be controlled, for example, with canary deployments, blue-green releases, and feature toggles [6].

Micro frontends reduce the technical risk further by allowing the frontend presentation layer to be broken down to the module level. This enables modules to be built, tested, and deployed independently of one another and the technology they use, minimizing the chance of cascading failure. It also allows teams to move faster with development without affecting the entire application stack [6].

Anti-corruption layers (ACL) are used to prevent legacy data and protocols, as well as legacy semantics, from polluting modern services. By normalizing the legacy interface to a canonical domain representation through the ACL, ACLs can help prevent legacy architectural decisions from leaking into newly built services and amassing technical debt. This process supports clean architectural evolution and controlled modernization in heterogeneous enterprise environments [6].

3.2 Operational Risk Containment

Operational risks. There is the risk of failed roll-outs, outages, inadequate scalability and lack of necessary incident recovery mechanisms for legacy system modernization. In the big-bang version of legacy system migration, most operational risks are concentrated in one event triggering an increased probability that service downtime is prolonged, service quality may degrade and system-wide cascading failures may occur.

In contrast to the incremental modernization pattern, which distributes operational risk between multiple iterations, and the strangler fig migration pattern, which allows the legacy system and the modern solution to co-exist, with traffic to production being gradually transferred to the portion that has been modernized, using this approach may also limit the blast radius of any failures, in that partial failures are contained, managed, and recovered from.

Micro frontend architectures are also shown to provide additional operational resilience through modular decomposition, independent deployments and autonomy among teams. Empirical evidence for the operational and organizational benefits of micro frontends, as well as adoption rates, is documented in academic literature on the topic. Support for heterogeneous frontend technologies is the feature that is most commonly cited as important across all sources (51.16%), and permits organizations to adopt any appropriate framework without lock-in. Cross-functional autonomous teams (41.86%) lead to smaller, faster delivery and higher ownership. Reduced coordination and deployment coupling between teams was the most common reason for using an independent development, deployment and release pipeline (34.88%) followed by potential speed and quality improvements, improved scalability (11.63%), testability (9.3%), fault isolation and resilience (6.98%), onboarding speed (6.98%), performance (4.65%), future proofing (4.65%) and fast initial loading (2.33%) [4].

Anti-corruption layers can also reduce operational risk by insulating modernized components from variant behavior in inconsistent legacy components, and translating between and encapsulating their legacy dependencies. This is a monitored, rate limited and isolated access point for a larger service and adds a level of resilience to the system and an architectural boundary that reduces the chance of failure propagation. This allows the rollback and containment effort to be focused, reducing MTTR.

3.3 Business Risk Management

From the business perspective, risks are a main concern to be avoided in a legacy modernization effort, to avoid negative impacts on business continuity, business strategy and stakeholder confidence. Risks include business disruption, loss of institutional knowledge, integration issues with other IT systems, and project budget overruns [7].
Incremental modernization can help organizations manage the business risks intrinsic in data center modernization. By breaking the project up into incremental phases, organizations can validate the new functionality against actual users in a production environment, reducing uncertainty, allowing business leaders to deliver business value, and enabling them to adjust their plans with feedback from stakeholders. Early incremental delivery of business value and realization of results builds confidence from stakeholders in the modernization roadmap and increases the likelihood of continued investment in the initiative [7].
Additionally, quality business continuity planning throughout the modernization effort helps reduce risk by ensuring that fallback positions are available and that there is never undue reliance on unproven solutions. This requires careful planning and integration, as well as active monitoring during the migration and transition activities, in order to provide business value and stability from the new solutions [7].

4. Quality and Stability Outcomes

Organizations adopting micro frontends see a temporary increase in defect density as teams learn the new architecture pattern and how to adopt the independent delivery approach. However, defect density tends to decrease thereafter as teams learn to deliver small, bounded changes more consistently, which helps to diagnose the problem within a bounded modular subsystem rather than the monolith. This is backed by industry data, which shows when failures are isolated to specific micro frontends, teams can often fix issues 30-50% faster than in customary monolithic patterns [8].

1. Localized Failure Containment: Since micro frontends are encapsulated and a failure in one domain does not affect the other domains, the blast radius of failures can be minimized and the application can remain stable in production even when the individual components are continuously evolving independently of one another [8].
2. Improved Stability over Time: As cross- team engineering practices improve, deployments become more frequent with at least as good stability outcomes, in part because each micro frontend can be deployed or rolled back independently from the other micro frontends, reducing the frequency of system outages [9].

5. Operational Evolution

Similar to software modernization, micro frontends enable change in engineering operations by allowing teams to operate independently to speed up delivery pipelines, decreasing inter-team coordination costs when deploying new code, and allowing teams to choose their own technology within architectural guardrails appropriate to their domain. This organizational shift enables:

1. Autonomous, Parallel Delivery: Teams can independently develop, test and deploy changes in their respective micro frontends, which reduces the coordination overhead and leads to greater throughput across different domains [10].
2. Better operational practices: Separation of concerns increases the ability to deploy, observe, and gather feedback on the completed system, based on production rather than design and build data [10].

Table 1: *Evolution of Quality and Stability in Micro Frontend Adoption [8] [9]*

Aspect	Initial Phase	Mature Phase	Key Mechanism
Defect Density	Temporary increase as teams learn new architecture pattern and independent delivery approach	Decreases as teams master delivering small, bounded changes consistently	Problem diagnosis confined to bounded modular subsystem rather than entire monolith
Failure Containment	Learning to work with encapsulated components	Failures isolated to specific micro frontends without affecting other domains	Blast radius minimized through domain encapsulation, allowing continuous independent evolution

Issue Resolution	Adaptation to new debugging and troubleshooting processes	Faster fixes due to isolation within specific micro frontends	Clear boundaries enable targeted problem identification and resolution
System Stability	Initial adjustment period with new deployment patterns	Improved stability outcomes with more frequent deployments	Independent deployment and rollback capabilities for each micro frontend reduce system-wide outage frequency
Team Practices	Teams learning architectural boundaries and autonomous delivery	Cross-team engineering practices mature and standardize	Enhanced collaboration patterns emerge while maintaining team autonomy

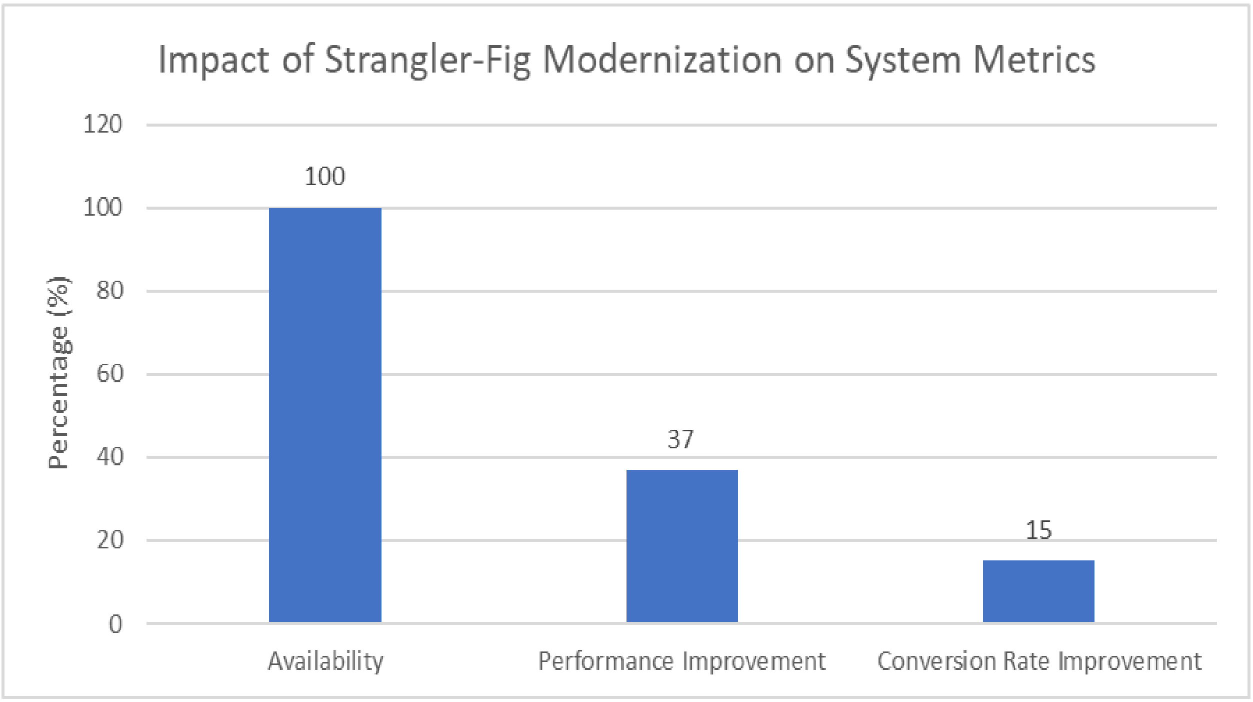

Figure 1: *Impact of Strangler-Fig Modernization on System Metrics [3]*

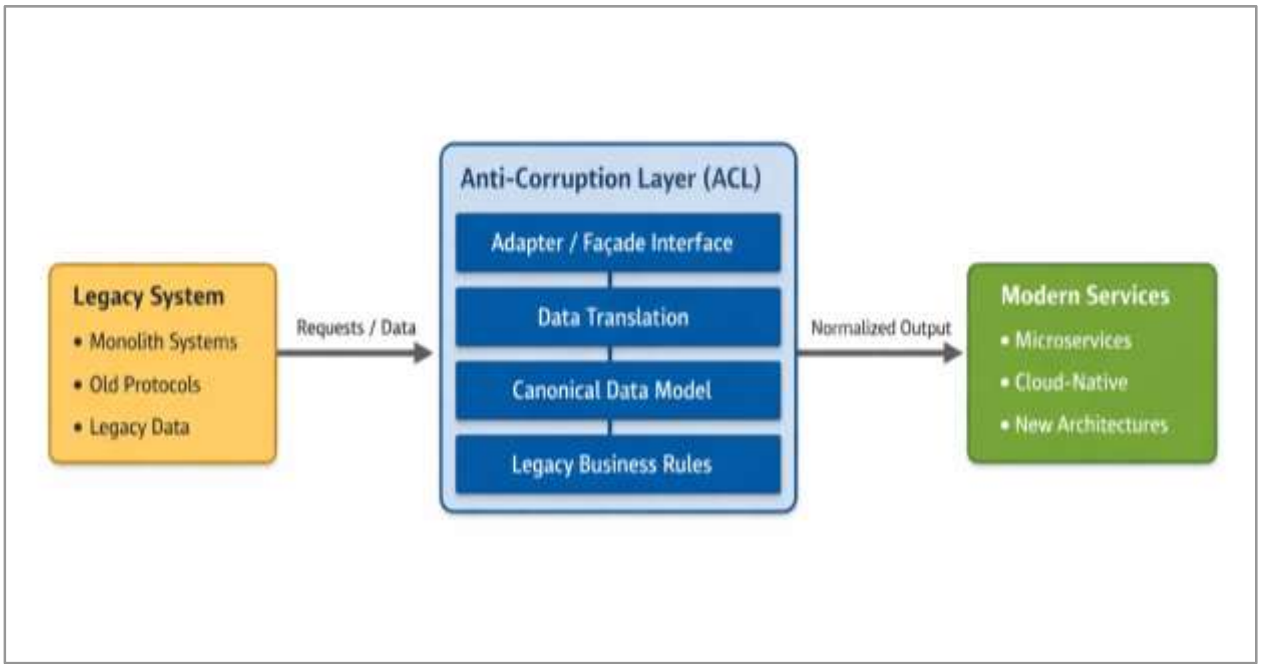

Figure 2: *Understanding Anti-Corruption Layer in Legacy Modernization [5]*

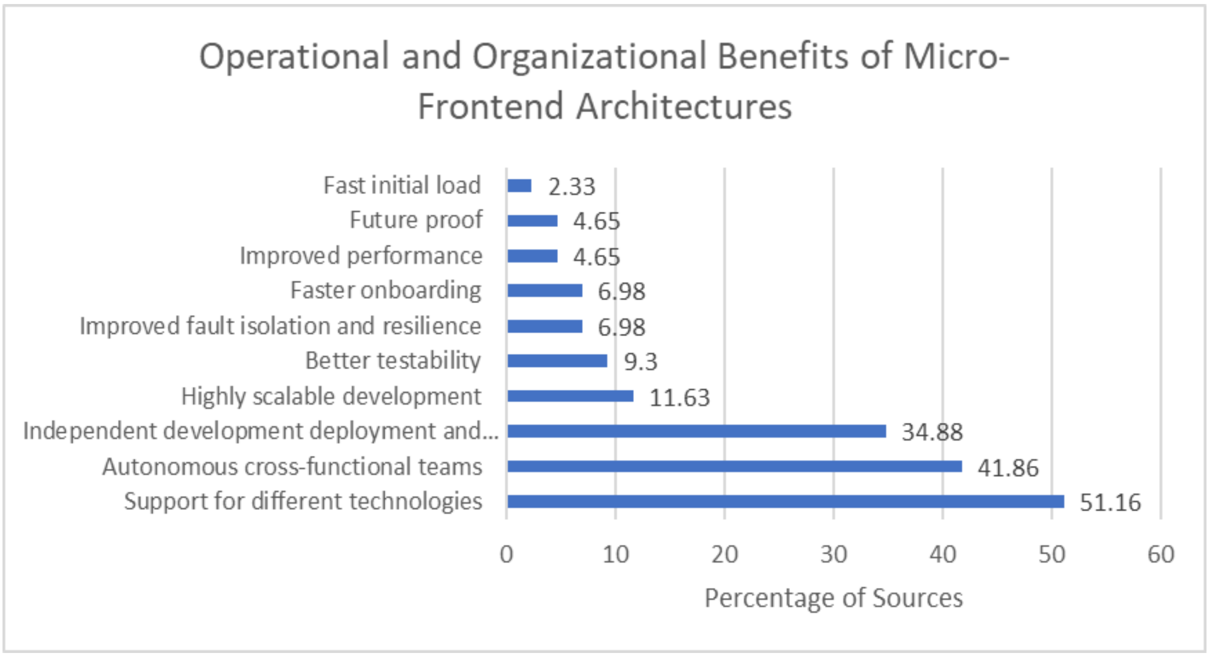

Figure 3: *Operational and Organizational Benefits of Micro-Frontend Architectures [4]*

6. Conclusions

Within the context of a micro frontend architectural approach, the Strangler-Fig Pattern provides an empirical model for organizations to modernize their legacy core applications by decoupling and modularizing the monolithic portions of a core application and incrementally replacing them while creating minimal operational risk to the underlying platform and creating maximum stability for the platform and the business. Since each micro frontend has its own functionality, issues often are present only in the micro frontend. These problems are easier and faster to fix. Modular components improve software quality because components can be tested and verified in iterations and continuous feedback cycles to the migration step.

Beyond the technical benefits, this encourages organizational and operational change. Teams associated with a domain have full autonomy to make technology and deployment choices and hold full end-to-end responsibility for how their services behave and are deployed. Knowledge silos dissolve as product teams' deep domain expertise is developed. Interdependencies are mapped and integration points are found. Deployment frequency increases and the severity of incidents decreases, thereby increasing delivery throughput and reducing risk.

For those businesses with large, mission critical platforms, this incremental evolution provides a practical approach to software modernization and software development that moves relatively large, risky and speculative rewrites into the domain of incremental evolution. While maintaining agility, operational performance and resilience, it allows a modular architecture to be built to adapt to changing business and technology requirements.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.
- **Use of AI Tools:** The author(s) declare that no generative AI or AI-assisted technologies were used in the writing process of this manuscript.

References

[1] Declan Good, "Legacy Transformation," in Technology Research Club, 2022. [Online]. Available: https://semanticdesigns.com/Company/Publications/Legacy%20Transformation.pdf

[2] Gatner, "Enterprise Resource Planning to Optimize Operations," Information Technology, 2020. [Online]. Available: https://www.gartner.com/en/information-technology/topics/enterprise-resource-planning

[3] Richard Gross, "Strangler Fig Pattern," Maibornwolff [Online]. Available: https://www.maibornwolff.de/en/know-how/strangler-pattern/

[4] Severi Peltonen, et al., "Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review", in Science Direct, Information and Software Technology, Volume 136, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000549

[5] Microsoft Learn, "Anti-Corruption Layer Pattern", Architecture Center. Available: https://learn.microsoft.com/en-us/azure/architecture/patterns/anti-corruption-layer

[6] Ashwin Ramachandran, "Legacy System Modernization: How to Reduce Risk and Unlock Value", in Precisely, Data Integration. 2025. Available at: https://www.precisely.com/data-integration/legacy-system-modernization-how-to-reduce-risk-and-unlock-value/

[7] Anas Tawileh, et al., "Strategically de-risking legacy modernization to secure the business and drive efficiency," CGI. [Online]. Available: https://www.cgi.com/en/alliances/AWS/strategically-derisk-legacy-modernization-secure-the-business

[8] Iternforge, "Microfrontend Architecture: Evolving Frontend Strategies to Unlock Scalability, Velocity, and Performance" [Online]. Available: https://iterforge.com/microfrontend-architecture-evolving-frontend-strategies-to-unlock-scalability-velocity-and-performance/

[9] Abhishek Pareek, "Future-Ready Frontends: Building Scalable Enterprise Applications with Micro Frontend Architecture", Developer.Dev, 2026. [Online]. Available: https://www.developers.dev/tech-talk/future-ready-frontends-building-applications-with-micro-frontends.html

[10] Erez Eizenman, et al., "Need micro frontend benefits at scale? Reimagine the operating model", in MCKinsey & Company, 2026. [Online]. Available: https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/tech-forward/need-micro-frontend-benefits-at-scale-reimagine-the-operating-model?utm_source=chatgpt.com