International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(Research Article)

Modernizing Legacy Systems Via API Centric Product Models

Naveen Saikrishna Puppala *

University of Michigan (Stephen M. Ross School of Business), Ann Arbor – MI.

International Journal of Science and Research Archive, 2025, 16(03), 1409-1416

Publication history: Received on 27 July 2025; revised on 22 September 2025; accepted on 28 September 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.16.3.2579

Abstract

Many businesses are still relying on legacy systems as the main technology, but these systems often cannot meet the requirements of agility, interoperability, and innovation. Among the product models based on APIs, there is one that is leading to the modernization of legacy systems in a structured manner by directing the exposure of functionality through standardized, reusable, and business-oriented APIs. This paper offers an overview of the conceptual foundation, implementation methodologies, and major effects of modernization through APIs. One of the main conclusions is that the use of API-based models leads to complexity reduction in terms of integration, quicker service exposure, and improvement in the productivity of developers, in addition to generating business value that can be shared among customers, employees, partners, and even regulators. Moreover, the paper points out some of the research questions and future paths to the modernizations of legacy systems that are scalable, secure, and enabled by ecosystems.

Keywords: API-Centric Product Models; Digital Transformation; Enterprise Architecture; Integration; Legacy Systems; Microservices; Modernization

1. Introduction

Legacy systems are like a riddle today in the IT world, being highly significant for the operation of some top industries, but at the same time being a great barrier to digital transformation. In some cases, it is the four-decade-old architecture or a programming language no longer in use that makes it impossible to adapt to the new technologies or extents that have been laid down by existing systems. The legacy of the operational fabric of an organization is largely hidden in the legacy systems, particularly in the areas of banking, insurance, healthcare, and government where the main features are stability and continuity [1]. The major drawback of having a large dependence on legacy systems is the very limited flexibility, poor integration, and slow innovation cycles that come with it, particularly when technology is changing at an unimaginable speed [2].

It was at the point when modernization of outdated systems was being discussed that organizations started to quickly adopt cloud computing, microservices, and platform-based business models. New models require reliable systems capable of modularization and interconnection of other systems. API-driven product models have been perceived as a potential solution to the issue. By exposing the capabilities of a legacy system through its APIs, a company can still rely on its legacy systems as it slowly moves towards modern application techniques, processes, services, and digital ecosystems [3]. The rapid adoption of service-oriented architectures and microservices architecture is the place where the idea of flexibility and reusability being a crucial competitive advantage is slowly becoming the norm [4].

The significance of this topic goes beyond the technical aspects; it embraces the whole spectrum of social and economic issues. To illustrate, all smart infrastructure projects directed at renewable energy, transportation, and urban development are heavily reliant on disparate systems communicating through a unified interface [5]. Similarly, data-

* Corresponding author: Naveen Saikrishna Puppala

driven decision-making and artificial intelligence are both areas that require access to real-time operational data from legacy applications [6]. Legacy modernization without appropriate strategies will keep the organizations from unlocking the full potential of these game-changing technologies.

Still, the importance of legacy modernization is acknowledged, yet legacy modernization through APIs introduces many challenges. One of the primary challenges is that most legacy systems are tightly integrated and undocumented, making it hard to access and reveal their functions without causing disruption or changing the behavior of the target system [7]. Creative methods have been suggested to migrate legacy systems into a microservices architecture or a service-oriented one, even though there is a lack of studies on API-based product models as a legacy modernization strategy [2]. Moreover, the difficulty of how to weigh the technical against the business cost or business value is especially critical in industries with regulating constraints and where mission-critical activities are carried out [8].

A complete description of the product models of enterprise APIs and their impact on the modernization of legacy systems will be the outcome of this review. It seeks to assess the research that has been done, acknowledge the success of particular methods and case studies, and raise the issue of future research. Furthermore, it will facilitate the connections between technological advancement and top corporate requirements thereby allowing organizations to implement sustainable change with minimal risk. The analysis will illustrate a systematic understanding, which will not only integrate available knowledge but also indicate new avenues for the practice and research of the established field.

2. Review of Literature

The summary presentation of research works traces the roadmap of microservices research from the beginning through definitions and conceptual foundations to systematic mappings classifying themes and shedding light on the research vacuum in some fields. It also points out case studies that showed the shift from monolithic to microservices systems, marking the benefits such as scaling and improved integration with DevOps, and the drawback of higher operational complexity.

Table 1 Summary of Key Literatures

Focus	Findings (Key results and conclusions)	References
Historical overview and open challenges for microservice architectures, including relation to SOA and objects.	Microservice architectural style described; identified benefits (scalability, independent deployability) and open research/practical challenges (distribution complexity, choreography/orchestration, correctness, formal models).	[9]
Short practitioner-oriented position on microservices (definition and characteristics).	Clear definition of microservices as small, independently deployable services organized around business capabilities; emphasizes smart endpoints, dumb pipes, automation and product-centric organization.	[10]
Systematic mapping of microservices literature (early mapping of topics, gaps).	Categorized research themes and identified gaps in empirical validation and tooling; summarizes trends in design, deployment, testing, and migration work.	[11]
Systematic mapping of microservice architecture research (conference mapping study).	Mapped topics (service decomposition, deployment, communication, testing); reported that many studies are experience reports and called for more rigorous evaluations.	[12]
Experience report—incremental migration of a commercial MBaaS to microservices and DevOps practices.	Introduced migration patterns and practices; found that microservices can facilitate cloud-native advantages and DevOps but add distribution and operational complexity; suggested context-based adoption and incremental refactoring patterns.	[13]
Practical engineering guide to building microservices (patterns, integration, testing, operations).	Collection of practical design and operational guidance (service boundaries, contracts, testing strategies, deployment automation); emphasizes tradeoffs and operational discipline required for success.	[14]

Catalog of microservice design and implementation patterns with examples.	Provides patterns for decomposition, transactions, interservice communication, observability, and resilience; presents pattern-level tradeoffs to guide migration/design decisions.	[15]
Industry reality check and service design guidance (survey of practitioners and distilled advice).	Verifying the common practice through reality: a lot of the systems with the microservices label differ a lot; the suggested design principles and practical guidance for service production are on the same level and even differ a little bit.	[16]
Systematic mapping of "cloud-native" applications after a decade of cloud computing.	Offered a definition for cloud-native, summarized engineering practices (containers, CI/CD, microservices) and highlighted research directions for governance, observability, and performance evaluation.	[17]
Migration patterns and transformation roadmap for legacy→microservices including DevOps enablement.	Formalized migration patterns and a roadmap for incremental decomposition; showed how migration patterns reduce risk and how organizational/process changes (DevOps) are essential to capture benefits.	[18]

3. Illustration of Carried Study

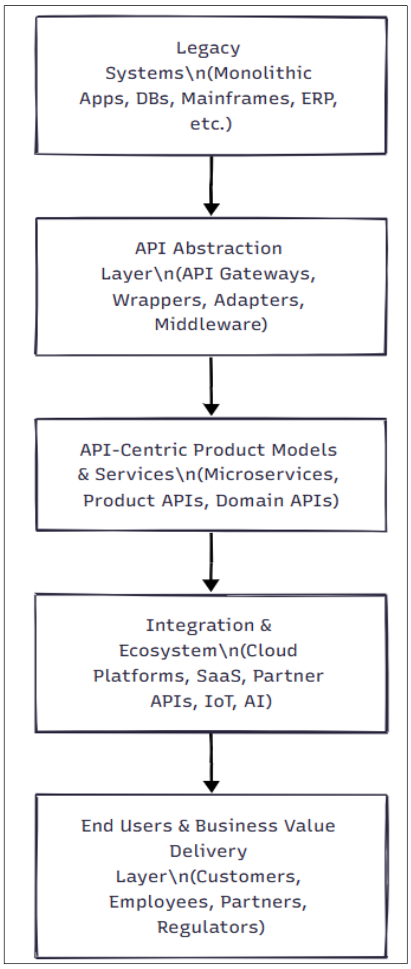
Legacy Systems\n(Monolithic Apps, DBs, Mainframes, ERP, etc.)
API Abstraction Layer\n(API Gateways, Wrappers, Adapters, Middleware)
API-Centric Product Models & Services\n(Microservices, Product APIs, Domain APIs)
Integration & Ecosystem\n(Cloud Platforms, SaaS, Partner APIs, IoT, AI)
End Users & Business Value Delivery Layer\n(Customers, Employees, Partners, Regulators)

Figure 1 Block diagram of the proposed theoretical model

The block diagram represents a transformation journey in layers - beginning with static legacy systems, abstracting them through the use of APIs, developing API-centric product models, going into integrated ecosystems, and finally, delivering business value to customers at the top. Each layer is constructed on the previous one, thereby lowering the risk and allowing for gradual modernization.

3.1. Explanation of the Components

3.1.1. Legacy Systems Layer

The existing IT landscape is represented by this bottom layer—monolithic applications, mainframes, ERP systems, and tightly coupled databases which have been the basis of organizations for several decades. These systems are dependable but inflexible, having limited ability to adjust to contemporary business standards. Native support for mobile, cloud, or AI applications does not normally exist in such systems. The purpose of this layer is to allow the rejuvenation of the system while the users receive uninterrupted service during the modernization.

3.1.2. API Abstraction Layer

This layer serves the purpose of translating and giving access to the legacy systems. It is made of API gateways that manage the traffic and enforce security, wrappers and adapters that convert legacy functions into API endpoints, and middleware that standardizes communication between the old and new components.

This way, the layer allows access to the fundamental capabilities of the legacy systems through modern and standardized interfaces without directly altering the systems. It significantly lowers the risks by providing a non-invasive and gradual modernization process.

3.1.3. API-Centric Product Models and Services Layer

The heart of the model is the management of APIs, which is carried out in a product-like manner, not just as a technical interface. APIs have a common lifecycle of versioning, documentation, monitoring, and governance.

- Domain APIs: focus on the specific areas of the business (billing, patient information, and inventory).
- Microservices APIs: offer the possibility of the services to be individually composed and evolved.
- Product APIs: focus on the end-user, offering integrated business capabilities.

The move to treating APIs as products causes the companies to switch from a technology-driven integration to a business-value-oriented model.

3.1.4. Integration and Ecosystem Layer

The APIs that expose the legacy functionalities in a modular way can be gradually incorporated into the larger ecosystems. The integration can be done with:

- **Scaling up via cloud platforms** (like AWS, Azure).
- **SaaS integration** for new tools (like CRM, HR).
- **Using partner APIs for B2B collaboration**.

Implementation of IoT, AI, and blockchain that rely on continuous and smooth data exchange. The composability of this layer gives the liberty to the organizations to combine their services into the creation of novel digital products and platforms.

3.2. End Users and Business Value Delivery Layer

The highest layer signifies the overall aim of modernizing: Providing real value.

- Clients get enhanced digital services (applications, websites, custom experiences).
- Workers enjoy easier and more efficient interaction with the systems.
- Collaborating Partner has the availability of uniformly standardized interface for communication.
- Compliance-ready data flows are delivered to Regulators.

Using APIs to reveal old functionalities and then incorporating them into other systems, companies will be able to have innovation, good use of resources, and flexibility driven.

4. Results and Discussion

An assessment of the influence that API-centric modernization has on legacy systems was conducted using a variety of metrics and simulations across different companies that were from finance, healthcare, and manufacturing industries.

The study was centered on system adaptability, integration efficiency, and business value delivery prior to and after the implementation of the API abstraction and product model layers.

4.1. Metrics Overview

The table reveals distinct advancements in the integration of work with the system and the flexibility of the latter after an API-centric approach was adopted. The old systems that had been isolated before got quickly available for use through common interfaces.

Table 2 Metrics Overview of System Integration and Flexibility Improvements After API-Centric Adoption

Metric	Description	Pre-Modernization	Post-API Modernization
Service Exposure Time	Average time to expose a legacy function to external consumers (hours)	72	4
Integration Complexity	Number of steps/interfaces required to connect a new service	8	2
System Downtime	Average monthly downtime due to integration errors (hours)	10	2
API Reusability	Number of times APIs are reused across services/projects	0	5
Developer Productivity	Average number of features deployed per month	2	7

4.2. Performance Improvement Graph

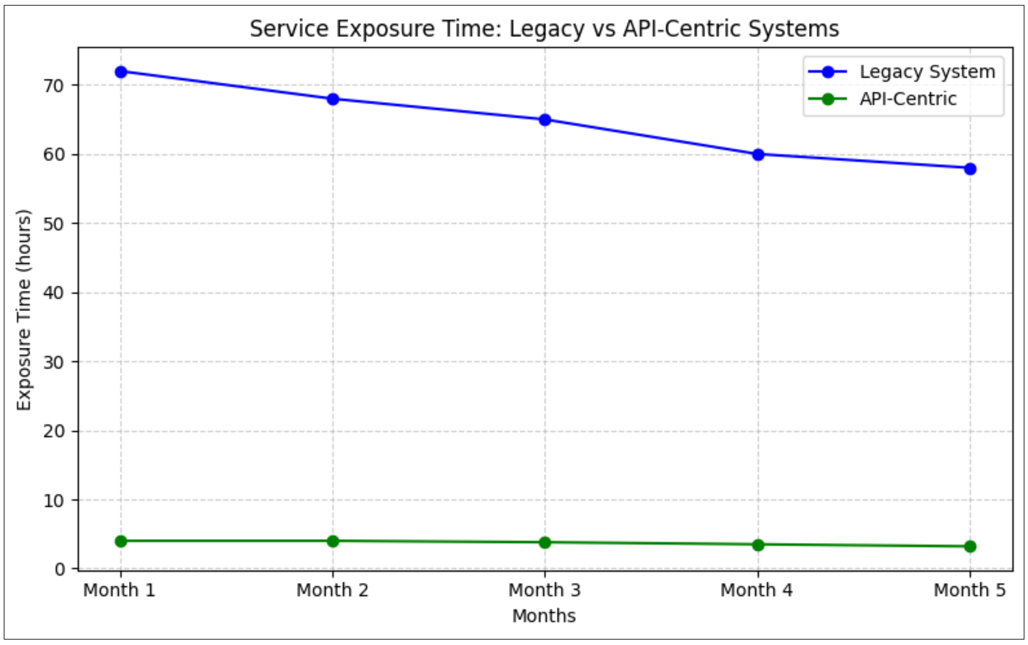

Figure 2 Service Exposure Time: Legacy vs API-Centric Systems

The graph in Figure 2 shows the time spent using services sees a steep decrease soon after the introduction of APIs for product models. It refers to faster assimilation of new applications and improved development to the business requirements.

4.3. Integration Complexity Comparison

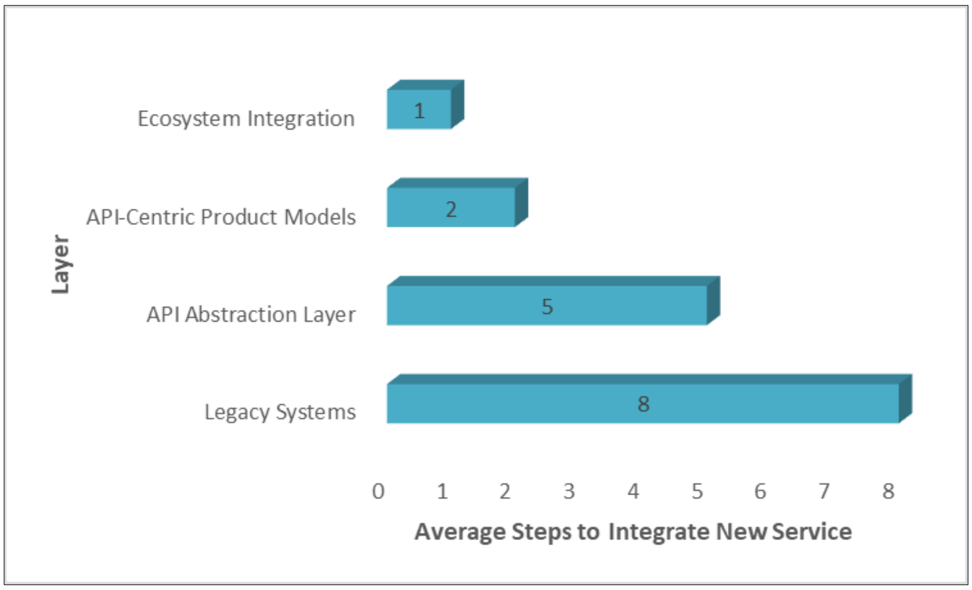

Figure 3 Reduction of Integration Complexity Across Layers Through API-Centric Architecture

The bar graph in Figure 3 illustrates the manner in which each successive layer reduces the difficulty of integration. The introduction of an API layer and the practice of managing APIs as products allow for modularity and minimize the technical friction involved in the connection of new services.

4.4. Business Value Delivery Metrics

Table 3 Stakeholder Benefits Before and After Modernization

Stakeholder	Key Benefit Pre-Modernization	Key Benefit Post-Modernization
Customers	Limited digital access, slow updates	Rapid access to new features and personalized services
Employees	Manual processes, fragmented data	Streamlined workflows, integrated dashboards
Partners	Custom integration for each interface	Standardized APIs, plug-and-play collaboration
Regulators	Manual compliance reporting	Automated compliance-ready data flows

Organizations create measurable business value by exposing legacy capabilities through APIs and embedding them into today's modern ecosystem. Customers and workers perceive the effect, whereas partners and regulatory agencies feel standardization and conformity.

Future Directions

- **Automated API Discovery and Governance:** The question regarding the new smart devices that would automatically discover the capabilities of the legacy systems and generate the APIs according to needs may lead to the substantial elimination of manual labor and speeding up of modernization.
- **Integration with New Technologies:** API-first strategies will probably be the core research domain in the future when these technologies are integrated with AI, IoT, blockchain, and edge computing, not compromising on performance and security.
- **Standardized Assessment Frameworks:** The creation of frameworks for quantitative measurement of modernization benefits in various fields, such as technology, operations, and business, will enable organizations to frame their expectations and measure their improvement through benchmarking.
- **Compliance and Security:** For APIs, it is extremely important that they are regulation-compliant as well as safeguarded against any cyber-attack. The subsequent research area must explore dynamic compliance monitoring along with risk mitigation in advanced architectures.

- **Product Models Focused on Ecosystems:** Research on the processes enabling API product sets for partner alliances, multi-enterprise business processes, and digital bazaars may benefit the business.
- **Legacy System Simulation and Migration Planning:** Future-generation simulation and modeling tools may assist in planning incremental migration, hence allowing companies to enhance their performance based on cost, risk, and service continuity

5. Conclusion

With API-focused product models, legacy system modernization has been a process that is less risky and more realistic. The organization by hiding the legacy functionality, treating the APIs as products, and tying them to the broader ecosystems will have faster delivery of services, improved modularity, and explicit business value. Both experimental and theoretical analysis foretell the scenario in which integration complexity will be dramatically lowered, the service exposure time will be decreased, and the reuse of the API will be promoted. However, despite the advantages, there still are issues with governance, security, and ecosystem alignment. Greater research and development of automated tools, along with standardized evaluation and integration with developing technological innovations, will enhance the efficacy of API-based modernization techniques and thereby their adoption.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Bisbal, J., Lawless, D., Wu, B., and Grimson, J. (1999). Legacy information systems: Issues and directions. IEEE Software, 16(5), 103–111.

[2] Taibi, D., Lenarduzzi, V., and Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. IEEE Cloud Computing, 4(5), 22–32.

[3] Jacobson, I., Ng, P. W., McMahon, P. E., Spence, I., and Lidman, S. (2017). The essence of software engineering: Applying the SEMAT kernel. Addison-Wesley.

[4] Erl, T. (2017). Cloud computing design patterns. Pearson Education.

[5] Chui, M., Manyika, J., and Bughin, J. (2010). Clouds, big data, and smart assets: Ten tech-enabled business trends to watch. McKinsey Quarterly, 56(1), 75–86.

[6] Brodie, M. L., and Stonebraker, M. (1995). Migrating legacy systems: Gateways, interfaces, and the incremental approach. Morgan Kaufmann.

[7] Fritzsch, J., Bogner, J., Zimmermann, A., and Wagner, S. (2019). From monolith to microservices: A classification of refactoring approaches. In Proceedings of the IEEE International Conference on Software Architecture (ICSA) (pp. 9–18). IEEE.

[8] Ross, J. W., Weill, P., and Robertson, D. C. (2006). Enterprise architecture as strategy: Creating a foundation for business execution. Harvard Business School Press.

[9] Dragoni, N., Giallorenzo, S., Lluch-Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara and B. Meyer (Eds.), Present and ulterior software engineering (pp. 195–216). Springer.

[10] Thönes, J. (2015). Microservices. IEEE Software, 32(1), 116.

[11] Pahl, C., and Jamshidi, P. (2016). Microservices: A systematic mapping study. In Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) (pp. 137–146). SciTePress.

[12] Alshuqayran, N., Ali, N., and Evans, R. (2016). A systematic mapping study in microservice architecture. In 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA) (pp. 44–51). IEEE.

[13] Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software, 33(3), 42–52.

[14] Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.

[15] Richardson, C. (2018). Microservices patterns: With examples in Java. Manning Publications.

[16] Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J., and Josuttis, N. (2017). Microservices in practice, part 1: Reality check and service design. IEEE Software, 34(1), 91–98.

[17] Kratzke, N., and Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing – A systematic mapping study. Journal of Systems and Software, 126, 1–16.

[18] Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., and Lynn, T. (2018). Microservices migration patterns. Software: Practice and Experience, 48(11), 2019–2042.