International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I4P110

Volume 73 Issue 4, 72-78, April 2025

Original Article

From Monoliths to Composability: A Socio-Technical Analysis of Integration Complexities in Composable Enterprise

Shashi Nath Kumar

Software Architect, Florida, USA.

Corresponding Author : 2snku@gmail.com

Received: 11 March 2025 Revised: 12 April 2025 Accepted: 21 April 2025 Published: 30 April 2025

Abstract *- Enterprises are transitioning towards composability with a mix and match of commercially off-the-shelf Software and bespoke business software (through on-prem, cloud services or Software as a Service offering) popularized as MACH Architecture by MACH Alliance and Packaged Business Capabilities by Gartner. This strategy leads to selecting best-of-breed heterogeneous products, where system integration takes centre stage and has some significant socio-technical impact. This paper investigates the interplay of System Integration with critical organizational factors like governance, team structure, process alignment, skills, etc. It explores how managing API contracts, addressing diverse protocols and data formats, ensuring end-to-end observability, and implementing robust governance is crucial for mitigating the potential chaos of heterogeneous integration in composable ecosystems.*

Keywords *- System integration evolution, Packaged Business Capabilities, MACH, System integration, Enterprise integration, Composable enterprise.*

1. Introduction

The search for agility, rapid innovation and customer-centricity has led organizations to adopt composable Enterprise Strategies. This strategy demands assembling end-to-end business capabilities through the selection of modular “Packaged Business Capabilities” (PBC) [1] leveraging Microservices, API First, Cloud Native, Headless (MACH) [2, 3]. This allows organizations to select best-of-breed software components, including internal bespoke microservices, Commercially Of The Shelf (COTS), and external Software-as-a-Service (SaaS). The Commerce and Retail industry is spearheading this strategy, however, it has gained significant adoption in other industries such as Financials, Healthcare, Manufacturing and others.

This move from monolithic systems towards composability significantly increases the complexity of integration. The components may have vastly different integration requirements, protocols, data formats, security models, and levels of API maturity. Networking also plays a critical role as security requirements imposed by various compliances require varying degrees of secured private connectivity between on-prem and SaaS providers. Managing the interactions within this potentially chaotic landscape becomes a critical success factor in the organization.

The benefits of composable enterprise with MACH and PBCs are discussed widely, but there is a vast gap in addressing the specific socio-technical integration complexities. Existing research is focused on the technical aspects of microservices / APIs and the broader organizational changes required for agility; however, it overlooks the friction points specific to integrating diverse capabilities. This paper addresses this gap by analyzing the socio-technical dimensions of this integration complexity within MACH and PBC ecosystems. The core problem investigated is the conflicting expectations around the flexibility and the practical challenges faced while forming a multi-component, potentially multi-vendor ecosystem capable of deriving business value from the investments. It investigates the interplay between critical technology (patterns, standards, choices) and organizational (governance, team structure process alignment, skills development, and cultural adaptation) dimensions.

By analyzing these socio-technical dimensions, I have proposed that effectively managing this complexity requires robust technical solutions (like API gateways and event buses) and deliberate strategies for evolving the organizations beyond the technology factors and making an inverse Conway's Law manoeuvre.

2. Background and Literature Review

The evolution of software architecture represents an ongoing progression to manage complexity and accommodate accelerated change cycles. The N-tier architectures improved upon the separation of concerns provided by Client-Server models but often resulted in monolithic deployments with high internal coupling risky and big bang releases. Service-Oriented Architecture (SOA) aimed to improve reuse and integration; however, common implementations often suffered from centralized bottlenecks (Enterprise Service Buses), cumbersome standards, and persistent data dependencies. These limitations impede the velocity and adaptability essential for contemporary enterprises. Integration challenges were also inherent in these architectural paradigms. Integration was always a critical challenge in that paradigm. Although the MACH approach that promises Composable, Connected, Incremental, Open and Autonomous architecture provides increased flexibility through APIs, the rapid increase and heterogeneity of components further complicates the integration challenges and negates the proposed benefits. This necessitated proactive management to avert the formation of a distributed "Big Ball of Mud" [4] in the distributed ecosystem. The "Architect's Paradox" highlights the conflict between designing for perceived stability (correctness) and the inevitability of continuous evolution [5, 6, 7] that is amplified by the dynamic and heterogeneous nature of composable components. Similar complexities are encountered in the Networking and Infrastructure area to maintain a robust, scalable, observable and secure ecosystem with diverse deployment and security models.

3. Conceptual Framework

3.1. MACH Architecture

The MACH Architecture approach advocated by the MACH Alliance leverages four Key Pillars [9]:

3.1.1. Microservices

Business capabilities are realized by autonomous services that offer modularity and allow for independent deployment and scaling.

3.1.2. API-First

Decoupling and seamless integration are enabled by APIs as stable contracts. Essential components include API gateways and governance.

3.1.3. Cloud-Native

Fully leverage the features of cloud platforms to enable scalability, resilience, and automation (IaC, CI/CD).

3.1.4. Headless

Backend APIs can be consumed to build flexible omnichannel experiences, which are made possible by decoupling frontends.

3.1.5. Extendibility

This translates to the service's ability to be closed to modification but open to extension. It is often promoted, but not officially, as part of MACH Architecture.

These pillars form the MACH Principles that advocate for a Composable, Connected, Incremental, Open, and Autonomous ecosystem [9].

3.2. Packaged Business Capabilities

Gartner and industry analysts define PBCs as fundamental building blocks of a composable enterprise. They are self-contained, ideally autonomous, and exposed via API modules that align with well-defined business functions [8]. This modular structure allows organizations to assemble and reconfigure solutions to promote reusability and adaptability dynamically.

3.3. API-First as the Intended Solution

The MACH architecture's API-first principle directly addresses integration challenges. This is achieved by establishing APIs as reliable, well-documented contracts creating a standardized interaction method for components like microservices, PBCs, and frontends, irrespective of their internal workings. API Gateways and Schema Registries are essential in managing these interactions, potentially overseeing routing, security measures, rate limiting, and protocol/data transformations.

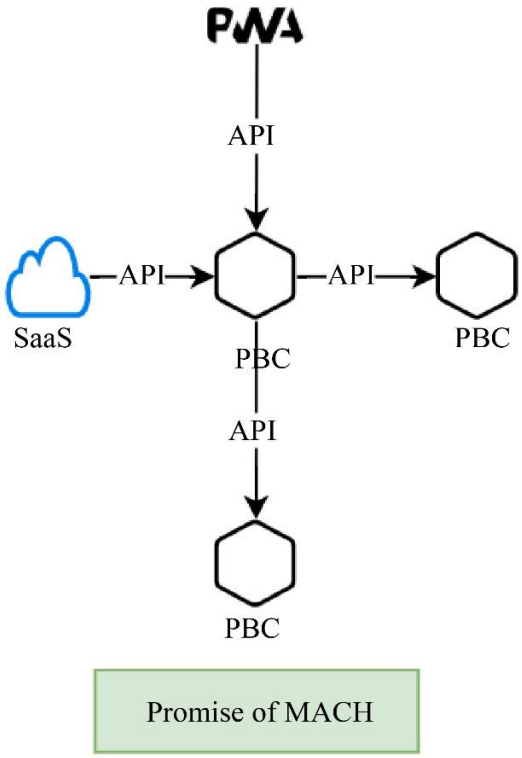

Fig. 1 Promise of MACH architecture: Composable and connected PBCs and SaaS

3.4. The Reality of Heterogeneous PBC Integration

The API-first principle of MACH guides the integration of PBCs. However, their heterogeneous nature has become immensely difficult to integrate. Irrespective of commercially off-the-shelf PBCs such as SaaS or internally developed bespoke PBCs such as Microservices, all have some common challenges while creating a homogeneous composable ecosystem.

3.4.1. Varying API Maturity / and Styles

There is a range of inconsistencies and issues in SaaS-based PBCs in their API diversity. The SaaS providers have different maturity levels in terms of quality and robustness. The design standards, coding practices, infrastructure selection, and platforms they use significantly affect their reliability. Their protocol selection also varies significantly. The newer SaaS vendors tend to invest in the pace of technical advancements, and no matter which infrastructure they use under the hood, they tend to accommodate some of the more recent protocols like GraphQL and Kafka, while more established players like to stick to hardened and battle-tested protocols like REST or MQTT. Practically, any new ecosystem needs some synergy with the existing legacy systems, which may still utilize legacy protocols like SOAP and even proprietary protocols. To accommodate this situation, a PBC ecosystem must support a variety of Synchronous and asynchronous protocols that require additional integration effort and expertise. This heterogeneity complicates the development and maintenance of SaaS PBCs and their seamless communication between the PBC and the external services.

The bespoke custom-built microservice APIs also exhibit inconsistencies due to various reasons. A microservices ecosystem needs strong standardizations and governance among the teams, as one of the core philosophies of microservices teams is independence. Since they are independent regarding technology and platform selection, this can lead to different architectural patterns, data formats, or authentication and authorization mechanisms. This internal inconsistency further complicates the development and integration process. Robust API management practices are required to address these challenges, including careful selection and evaluation of external APIs, strong governance and standardization of internal APIs, and appropriate tools and technologies to facilitate integration and ensure interoperability [10].

3.4.2. Diverse Data Formats / Models

Several incompatibilities between the data models and formats among the PBCs bring up the challenges of intricate data mapping and transformation processes during the integration. It can be due to adherence to different standards (like Cloud Events vs others) or message formats like XML, JSON, Avro or Protobuf. Anti-Corruption Layers (ACLs) are a standard way to mediate the interactions. However, they add complexity to the overall system architecture. Over time, these ACLs have become a complex integration ecosystem rather than staying true to their nature, and they need more capacity to design, implement, and maintain. On top of that, it introduces performance overhead because of the additional data transformation steps involved.

3.4.3. Different Security Models

The security model of PBCs varies widely. It can include OAuth 2.0 with various Grant types, SAML, API keys, basic authentication and custom tokens. This creates a complicated web of identity federation and token translation logic. The need for different Compliance standards makes the ecosystem further heterogeneous regarding security models. Some PBCs need more granular role-based data access, whereas others need more uniform one.

3.4.4. Asynchronous Integration Needs

Most SaaS provides synchronous APIs that are crucial but insufficient for creating a strong system integration as they introduce run-time dependency and uptime coupling. It is equally important to work with the available interface to integrate the PBC, and there should always be an attempt to reduce these couplings through asynchronous API communication [11]. SaaS platforms that offer Async APIs differ considerably in their support for event publishing and subscription. This inconsistency and limited support can cause integration issues, potentially requiring workarounds like polling or custom event connectors. Polling introduces latency and unnecessary system load due to frequent requests. Building custom event connectors is usually time-consuming and resource-intensive, requiring specialized skills.

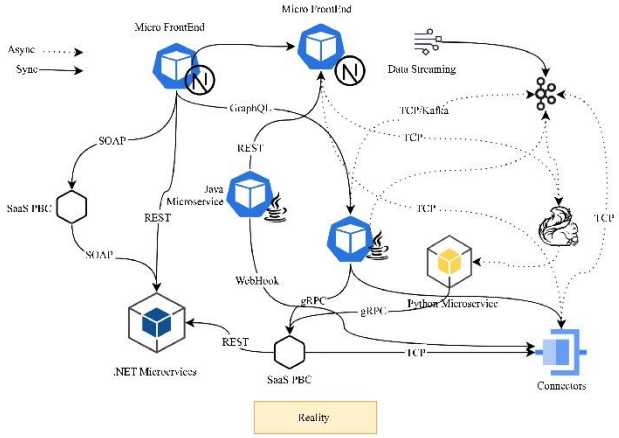

Fig. 2 The reality of a composable enterprise depicting the variety of formats and protocols

3.5. Socio-Technical Analysis

Managing this heterogeneity requires a multi-faceted, socio-technical approach beyond purely technical solutions. Several studies have shown that acceptance and confidence are crucial to adopting technology [12]. The following strategies are needed to address the complexity of integration.

3.5.1. Effective Governance and Standardization

API Design Standards

To enable smooth integration and interoperability among different PBCs, creating and applying a set of standardized API design guidelines is essential. These guidelines should include naming conventions, standardized error handling, defined versioning strategies, consistent authentication / authorization mechanisms, standardized data formats, and standardized development processes (e.g., templates or code generation) [13, 14].

Integration Patterns

Consistent and reusable integration patterns for common integration scenarios go a long way in addressing the heterogeneous integration requirements for PBCs. These patterns must be defined in the organization's scope and adopted uniformly. The typical example could be using an API gateway and or Service Mesh for synchronous communications but having separate patterns for internal and external APIs, usage of API portal, API design first, Open API Specification, usage of the type of messaging brokers in various scenarios like cloud-based messaging broker for external systems and on-prem broker, when to use persistent queue against when to use an in-memory queue, using schema registries with Async API specification.

Data synchronization patterns ensure consistency across multiple systems and avoid run-time data fetch. It is also crucial to balance standardization and flexibility to promote innovation and counter vendor lock-ins.

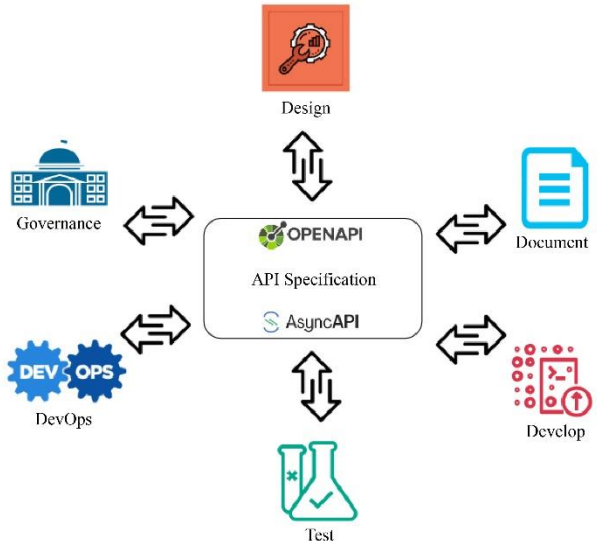

Fig. 3 Harmonization with API design standards and integration patterns where the api specification takes the centerstage

API and Event Contract Management

Robust process development and effective change management across diverse system components are required for managing API and event contracts. These processes should include strategies for versioning, automated (build pipeline embedded) consumer-driven contract testing for any breaking changes, and up-to-date documentation.

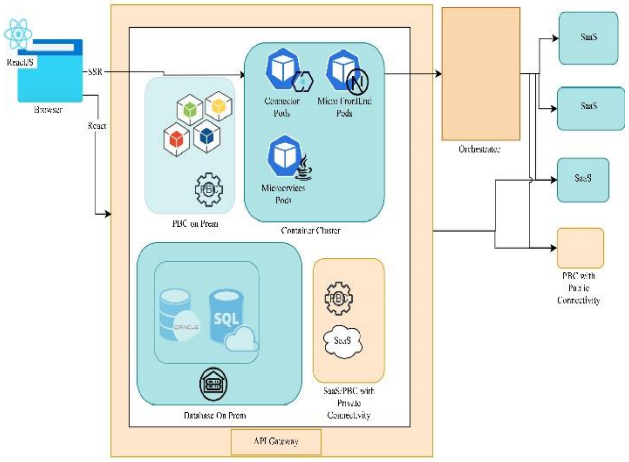

Fig. 4 Harmonization with api gateway and integration orchestrator

Governance Models

Organizations must choose between a centralized or federated governance model for managing their APIs and events based on their size, culture and specific needs to provide a common direction and uniformity to the development teams to manage the APIs' design, development, deployment and maintenance.

Additional Considerations

Uniform and robust monitoring and logging, rigorous and automated testing and quality assurance, integrated security practices, and architecting and designing for performance and scalability can be embedded in the patterns and governance.

3.5.2. Platform Engineering and Tooling

The internal platform team is key to streamlining the integration process for developers by providing the necessary tools and infrastructure [9]. They can enable development teams to self-service various infrastructures and follow patterns and governance guidelines.

3.5.3. Process Alignment and Evolutionary Integration

Integration strategies must align with agile processes:

Incremental Integration: Instead of implementing large-scale integrations simultaneously, consider adopting strategies like the Strangler Figure pattern or gradually introducing new SaaS PBCs alongside ACLs [16]. Fitness Functions for Integration: Automate tests to verify key integrations, API contracts, and end-to-end flows across multiple PBCs. These fitness functions will prevent regressions as individual components are updated [17]. Infrastructure as Code (IaC): Ensure consistency and repeatability by automating the configuration and deployment of integration components, such as ACLs, brokers, and gateways.

3.5.4. Culture, Team Structure and Skills

A successful strategy around team structure and skills is critical, and an inverse "Conway's Law" maneuver is needed for the organization. This requires careful reorganization of teams to reduce cognitive loads (align similar business function PBCs and technical stack) and build a culture of learning and transforming. It includes new strategies for building:

Integration Expertise: Integrating diverse PBCs, especially external SaaS platforms, requires specific skills beyond basic application development, including a deep understanding of various API styles, security protocol patterns, data mapping techniques, event-driven patterns, and specialized integration middleware. Targeted training, hiring, or specialized roles/teams are needed to build this expertise.

Cross-Functional Teams and Enablers: The teams managing end-to-end user journey must understand multiple PBCs and underlying Integrations. Management, Architects and Product teams are enablers to provide guidance and resolve impediments and conflicts [18].

Collaboration: Successful integration necessitates effective collaboration and communication between the teams responsible for different PBCs (internal or vendor teams). This includes clear communication about API changes, contracts, and shared responsibilities.

Cultural Transformation through learning: Adopting composable architectures requires a significant cultural shift towards fostering collaboration and breaking down silos, shared ownership and psychological safety and treating internal capabilities as products consumed via APIs. This requires learning from failures by blameless post-mortem, experimentation and sharing knowledge.

3.5.5. Managing Coupling in a Heterogeneous Landscape

While aiming for loose coupling, integrating diverse components introduces specific risks of data (use of PBC-specific formats), platform (SaaS/PBC provider) and temporal coupling (synchronous calls for queries) that requires a strategy to accept or mitigate the risks arising out of them.

4. Discussion

This analysis highlights that effectively managing integration complexity is a crucial socio-technical challenge. Achieving success goes beyond simply implementing an API Gateway. It necessitates strategic governance to define preferred integration patterns and standards, robust platform engineering to provide self-service integration capabilities, agile processes that enable incremental integration and validation through fitness functions, and teams with essential integration skills operating within collaborative structures. Navigating this landscape involves inherent socio-technical trade-offs that need further deeper investigation. For example, confident decisions regarding governance models (e.g., centralized vs. federated) present trade-offs between integration velocity and long-term architectural consistency that need careful consideration within heterogeneous PBCs. Similarly, the level of investment in platform engineering must be balanced against its impact on the cognitive load, required autonomy and various other factors.

The specific dynamics of managing multi-vendor SaaS ecosystems, including establishing inter-organizational trust and navigating conflicting vendor priorities, add another socio-technical complexity requiring deliberate management strategies. Failure to address these interconnected socio-technical aspects risks undermining the agility and flexibility promised by composability. Without deliberate management of integration complexity across technology, process, and people-including understanding the cognitive biases affecting decisions will inadvertently create a brittle, unmanageable distributed system-a modern manifestation of the "Big Ball of Mud" or the unrealized potential of earlier SOA initiatives

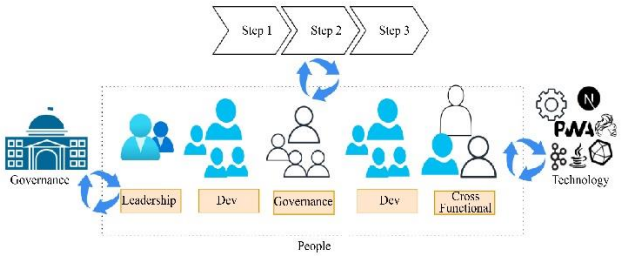

Fig. 5 Harmonization with the people at the core of the enterprise

Future research should focus on purely technical dimensions such as patterns for effectively governing heterogeneous API landscapes focussed on PBCs, strategies for managing data consistency across internal services and external SaaS PBCs, and developing better tools for testing and observing complex, multi-component workflows involving diverse integration points and a broader socio-technical dimensions.

Conducting longitudinal studies tracking organizations as they adopt and mature their composable architectures would provide valuable insights into how socio-technical integration challenges evolve and how organizational learning impacts success. Developing and validating metrics or models to quantify socio-technical factors such as team cognitive load relative to PBC integration complexity, communication overhead across boundaries, or the measurable impact of specific governance interventions will be a critical next step. More rigorous investigation into the socio-technical trade-offs inherent in architectural and organizational decisions within composable ecosystems is needed, potentially using case study or simulation methods. Focused socio-technical research is required to understand the complexities of

managing multi-vendor SaaS ecosystems, including trust dynamics, governance across organizational boundaries, and managing dependencies on external vendor roadmaps. Analyzing security technically and as an emergent socio-technical property within distributed, heterogeneous systems requires further study, examining how factors like distributed ownership, team structures, and organizational culture influence the overall security posture. Exploring how these socio-technical integration challenges manifest differently across various industries (e.g., retail vs. finance) and adopting composable approaches could yield context-specific insights.

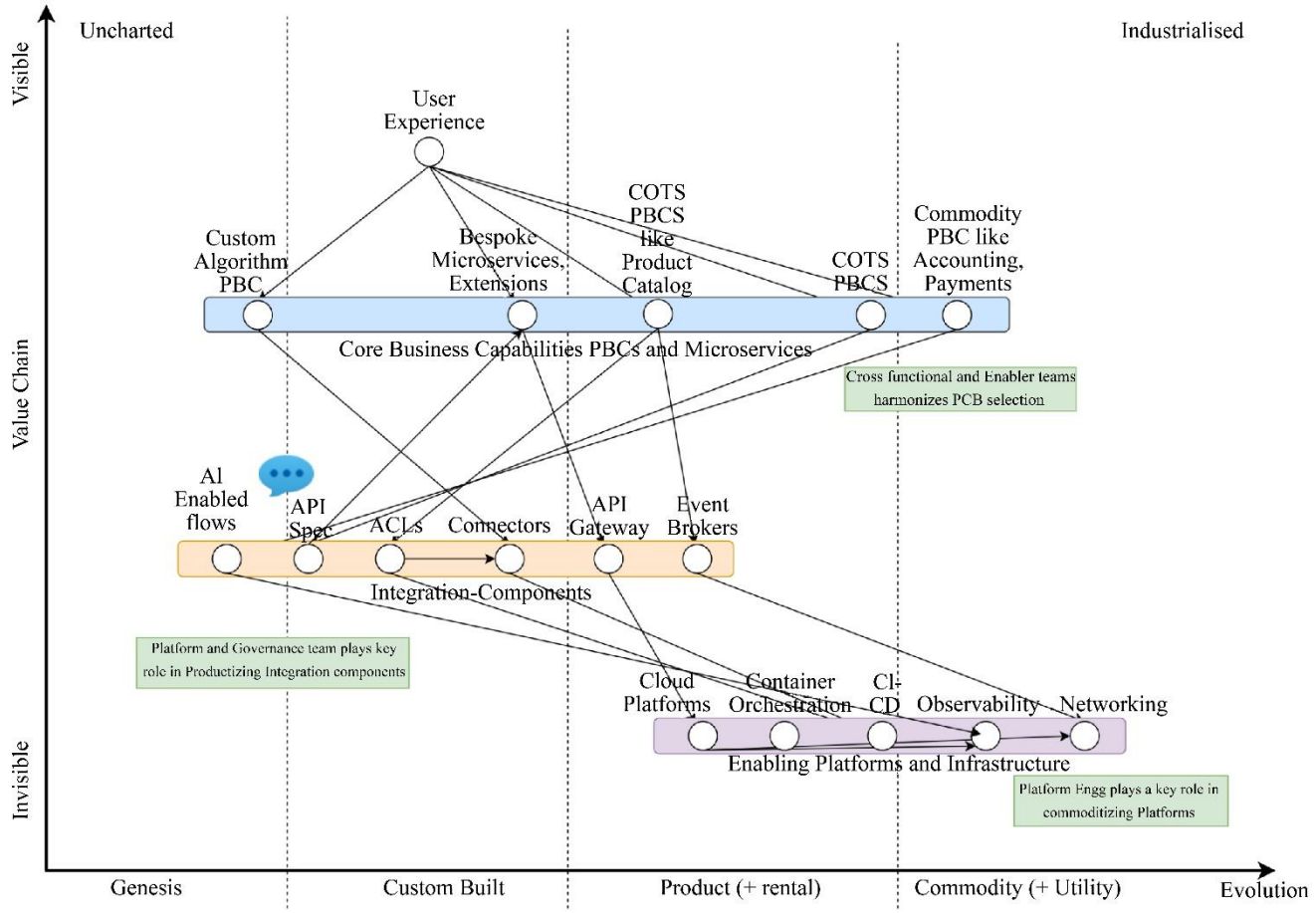

Fig. 6 Wardley map [19] illustrates the value chain for 'Rapid Business Capability Deployment' within the analyzed composable ecosystem. The Y-axis shows dependency from visible user needs (top) to invisible infrastructure (bottom). The X-axis depicts component evolution from genesis (left) to commodity (right). The map highlights the heterogeneity of components (e.g., bespoke microservices, COTS/SAAS pics, commodity infrastructure) and their complex interdependencies, illustrating the landscape where the discussed socio-technical integration challenges arise.

Further empirical studies examining the practical application and effectiveness of concepts like the Inverse Conway Maneuver and team cognitive load management within PBC adoption contexts would also be valuable.

Addressing these future research directions will contribute to a more comprehensive understanding of successfully navigating the transition to composable enterprise architectures.

5. Conclusion

The transition from monolithic architectures to composable enterprise strategies introduces not only technical challenges but also deep socio-technical complexities. As organizations adopt modular and best-of-breed components using MACH principles and Packaged Business Capabilities (PBCs), the need for seamless, secure, and scalable integration becomes paramount. This paper has illustrated that the diversity in APIs, data models, security standards, and communication protocols—combined with fragmented team structures and organizational silos—poses a significant risk of reintroducing the very chaos composability seeks to eliminate.

To mitigate these challenges, enterprises must adopt a holistic integration strategy that combines robust technical mechanisms such as API gateways, event brokers, and standardized contracts with strong governance models, platform engineering, and cultural transformation. Socio-technical strategies, including the inverse Conway's Law maneuver,

collaborative cross-functional teams, and investment in integration-specific skills, are critical to sustaining agility, scalability, and long-term architectural coherence.

The research underscores that successful composability hinges not just on choosing the right technologies, but on rethinking how people, processes, and platforms interact. Future studies should further investigate empirical metrics for socio-technical integration performance, case-based learnings on multi-vendor ecosystems, and the evolving role of governance in distributed digital enterprises.

References

[1] Composable Enterprise as Innovation Strategy, The MAMBU Website, 2020. [Online]. Available: https://mambu.com/en/insights-hub/articles/composable-enterprise-as-innovation-strategy

[2] Composable Commerce in 2025: Proven, Packaged, and Ready for the Mainstream, Composable.com. [Online]. Available: https://composable.com/insights/composable-commerce-in-2025-proven-packaged-and-ready-for-the-mainstream

[3] MACH Technology Explained, MACH Alliance. [Online]. Available: https://machalliance.org/mach-technology

[4] Brian Foote, and Joseph Yoder, "Big Ball of Mud," *Pattern Languages of Program Design*, vol. 4, pp. 654-692, 1997. [Google Scholar] [Publisher Link]

[5] O'Reilly, Barry, Architect's Paradox, On the Youtube Channel the Complexity Lounge, 2025. [Online]. Available: https://www.youtube.com/watch?app=desktop&v=Qq8x7KIV4W8&t=0s

[6] Barry O'Reilly, Software Architecture for a Rapidly Changing World, Boundaryless Podcast. [Online]. Available: www.boundaryless.io/podcast/barry-oreilly/

[7] Barry M. O'Reilly, "Residuality and Representation: Toward a Coherent Philosophy of Software Architecture," *Procedia Computer Science*, vol. 224, pp. 91-97, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[8] Sam Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*, O'Reilly Media, pp. 1-255, 2019. [Google Scholar] [Publisher Link]

[9] The MACH Principles, MACH Alliance. [Online]. Available: https://machalliance.org/mach-principles

[10] Open API Initiative Website. [Online]. Available: https://spec.openapis.org/oas/latest.html

[11] AsynAPI Initiative Website. [Online]. Available: https://www.asyncapi.com/docs/concepts

[12] Hamed Taherdoost, "A Review of Technology Acceptance and Adoption Models and Theories," *Procedia Manufacturing*, vol. 22, pp. 960-967, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[13] API Design Guide, Google, 2025. [Online]. Available: https://cloud.google.com/apis/design

[14] Vadake Narayanan, and Yamuna Baburaj, "Technology Standardization in Innovation Management," *Business and Management*, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Gregor Hohpe, Platform Strategy-Innovation Through Harmonization, The Architect Elevator. [Online]. Available: https://architectelevator.com/book/platformstrategy/

[16] Ian Cartwright, Rob Horn, and James Lewis, Patterns of Legacy Displacement, 2024. [Online]. Available: https://martinfowler.com/articles/patterns-legacy-displacement/

[17] Paula Paul, and Rosemary Wang, Fitness Function-Driven Development, 2019. [Online]. Available: https://www.thoughtworks.com/en-us/insights/articles/fitness-function-driven-development

[18] Gregor Hohpe et al., "The Software Architect's Role in the Digital Age," *IEEE Software*, vol. 33, no. 6, pp. 30-39, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[19] Wardley Maps. [Online]. Available: https://www.wardleymaps.com/