MACH Alliance

**2024**

# Interoperability

**PART 1**

# How to evaluate and integrate composable solutions

## (and what to do with non-conforming legacy systems)

The **benefits of composable business practices and architectures** for minimizing risk and maximizing agility are clear. The ability to rapidly add or change technologies are key to organizations moving to meet the ever-increasing demands on customer experience and **MACH principles (microservices, API-first, cloud-native SaaS, headless)** are foundational to driving interoperability between composable technologies.

So what are the key criteria and considerations organizations need to understand when looking at interoperability – and how can you also work within complex environments which may include legacy or on-prem applications?

## Business fit and interoperability fit

The primary consideration in evaluating composable technologies is to ensure that the application is fit for purpose. Beyond ensuring the solution fulfills business requirements, ***the level of interoperability and extensibility should also match that solution's importance to the business***.

For example, an organization in the consumer discretionary goods sector would consider commerce an "anchor" application. Because the ability to sell goods online is so critical to the business, this organization needs to integrate other systems around its eCommerce solution.

**These additional requirements would often include functionality like user interface extensibility and mature APIs. Extensible anchor solutions enable organizations to implement platform automation or embed interfaces from other lesser applications, providing a consistent and unified internal employee experience.**

## Stability and longevity

In the context of MACH, stability and longevity specifically refers to the entire composed system and practice around that rather than specific applications within your organization's digital estate.

Previously, organizations would make big bets that certain key systems would be in place for years (or even decades). In practice we have seen that the pace of software development has meant that companies that overly invested in vendors are now left with architectural debt from expensive and inflexible architectures. The MACH principles dictate that the overall approach to composability can enable functional stability and longevity.

In other words, despite similar names, ***stability is the exact opposite of being static***. An organization that has embraced MACH principles can swiftly adapt its systems with minimal disruption, enhancing agility, uptime, and reducing risk, unlike those hindered by legacy system constraints.

## Best-of-breed

One of the key advantages of buying best-of-breed software in a composable approach is that you no longer need to overbuy based on growth plans or vendor roadmaps. Following the principles of interoperability means that ***you can easily procure and integrate services as needed (and importantly, no sooner)***.

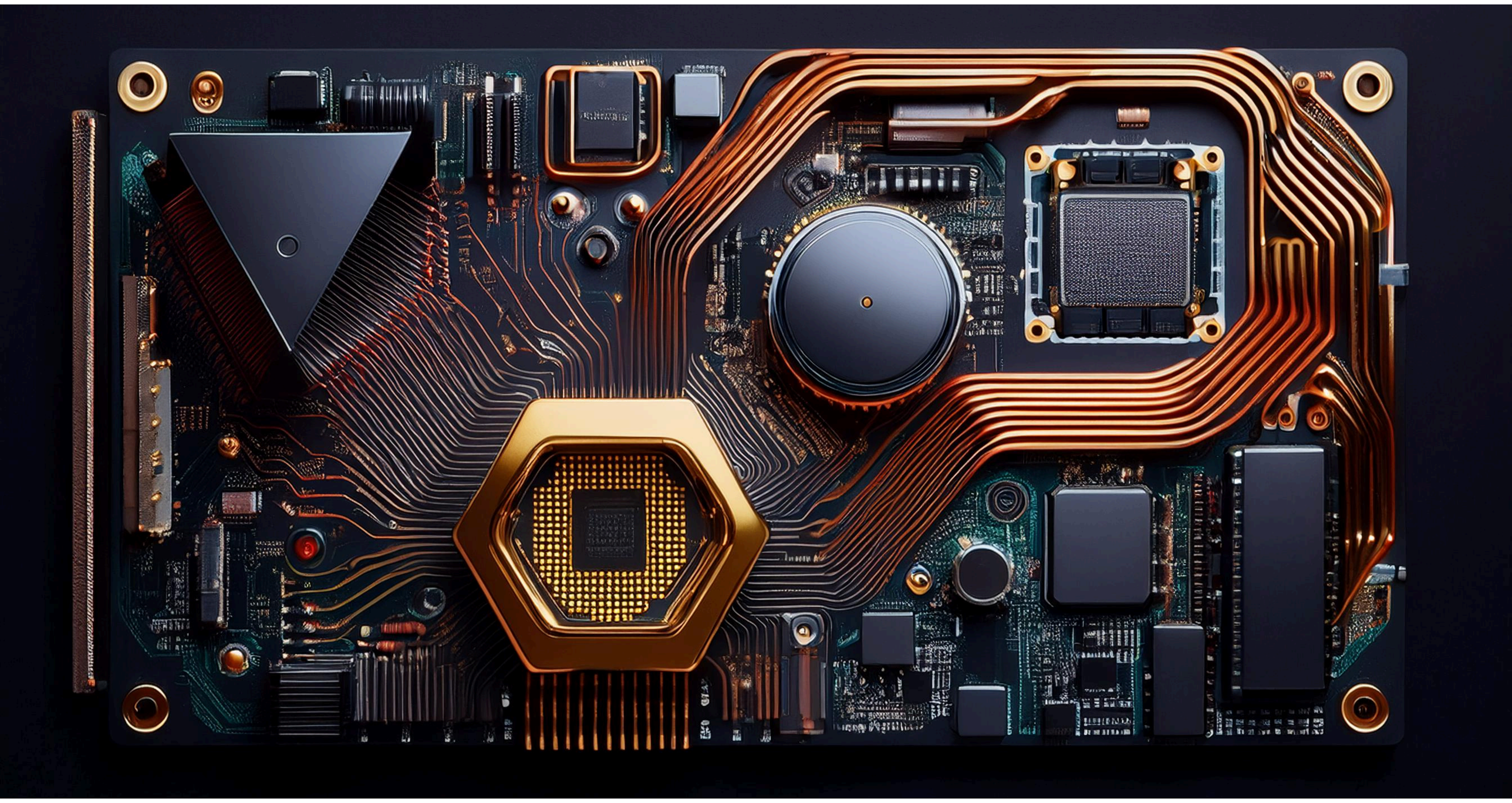
No longer are you dependent on vendor promises and roadmaps to align with long-term goals. Instead you can granularly pick and choose from multiple vendors based on technology that actually exists today and integrate them appropriately. If your other vendor comes along and builds that functionality a year or two down the road —great! It means you can look at potentially retiring your other acquired or built tool. The point is that you have the choice in the matter.

However, in order to have that choice means that your approach needs to adhere to the MACH principles—particularly those around granular microservices and headless APIs, in order to ensure that business functions can be separated out to the point of your requirements.

## No Vendor Lock-in

A modern system should enable multiple ways of working. This includes the ability to configure the SaaS application directly according to the organization's processes, as well as the ability to export these configurations.

This can enable agility between teams at the business and IT can work together to easily make changes, but also ensure their tasks and outputs are coordinated and backed up as part of normal operations. However, these enabling functions also mean that moving to other vendors or scaling across business units can be accelerated, as more of the application and configuration can be scripted for export and import functions.



## Modern architecture

These key pillars of business fit, stability, and flexibility are all interdependent on each other. A modern MACH architecture is built on these principles. However, the majority of organizations have existing systems that were built prior to these principles being commonly documented, understood and implemented.

*"So how can an organization ensure these two approaches work together?"*

# Common issues with existing legacy architecture

Given the longevity of some systems—especially those core to the business such as Enterprise Resource Planning (ERP)—working with older systems or formats is often a necessity, even in a mostly modern approach. Legacy architecture often has a few characteristics that were common at the time. These include:

## Older data formats

When working with legacy systems, developers are often dealing with older data formats that often lack the types of features and functions that are more common among modern composable systems.

For example, XML was once the most common way of doing data interchange. Because data interchange was often quite slow, XML had functions such as validation that could be used to ensure the correctness of payloads before they were sent. However, it has largely been replaced by JSON in many contexts.

The reasons for this change are somewhat related to JSON being able to easily parse into front-end applications, but mostly due to XML being overly structured and document-centric (which is less useful for microservices and granular content and events where more and smaller payloads are the norm). In other words, the features that made XML useful at the time are now considered more of a burden.

Of course, this is a fairly high-level example, but it extends into applications, file formats, operating systems and even transport protocols.

## Lack of event-based capabilities

One of the most common patterns within newer systems is the ability to trigger processes based on system events. In modern systems, these are most commonly implemented as webhooks which allow systems to communicate with each other in real-time.

Many older systems or APIs lack the means to signal other systems - meaning common tasks around indexing or synchronization of changes often come with some custom code or additional systems for polling changes.

## Lack of granularity

As discussed in the XML example above, it was appropriate at the time because often interchange between systems was done asynchronously (say, once a day) and working with a large, single document-based approach that could have validation, document hierarchies and element-based operations made sense.

Now that we have microservices and live eventing, these older (and heavier) means of data interchange are less suited for these applications.

## Poor performance and latency

Older systems were often designed based on requirements driven by hardware that was far less capable. As a result, data-intensive processes were often designed around asynchronous processes and large data transfers, rather than real-time eventing.

# How to approach integrating these systems into a modern stack

## Migrate in a staged approach

The approaches between a modern microservices and event-based approach have different needs relative to the older means of data interchange. So how can you adapt those legacy systems into these modern systems?

Instead of “big bang” migrations to completely rip-and-replace applications, the MACH approach (particularly the “microservices” focus) enables approaches such as the “strangler pattern” or “bubble context” where you can start to encapsulate existing legacy functions into smaller microservices and enable technological change in a staged approach.

Our MACH Alliance members have some resources on the topic:

- Commercetools has a useful high-level explanation of the strangler pattern here: **VIEW ARTICLE**

- AWS has a highly detailed technical explanation of the approach, covering examples services and key patterns such as domain-driven design (DDD) and anti-corruption layer (ACL) here: **VIEW ARTICLE**

## Intermediary layers

Another approach enabled by MACH technologies are vendors such as Conscia, Netlify (Connect), Occtoo, and Uniform, which provide a layer that can connect to underlying sources of content and data and expose them in a way that is more granular and event-based.

These systems can perform functions such as caching and calculation in order to specifically enable some digital experience scenarios such as building content experiences or personalization.

For more generic functions that may be related to back-end processing, ETL (extract, transform and load) tools can also be a valid approach for enabling a move to microservices.

MACH Alliance member Talon.one has an explanation here: **VIEW ARTICLE**

# Summary

Going back to some of the original research around composable business practices “**Gartner Keynote: The Future of Business Is Composable**” it becomes clear that composable technologies are key to enabling their four basic principles:

- More speed through **discovery**
- Greater agility through **modularity**
- Better leadership through **orchestration**
- **Resilience** through autonomy

By implementing composable techniques and technologies, organizations can adapt quicker (such as in the case of the pandemic and shifting delivery models or supply chains) or build differentiated customer experiences (mobile apps, omnichannel experiences such as click-to-collect).

Understanding your data flows and customer journeys are an important aspect to determine the needs for interoperability between those systems on which your business is built. Those key “anchor” applications such as commerce, content or customer data may require a greater level of interoperability as you evaluate their place within your composable architecture.

However, it is worth noting that when evaluating interoperability on the above criteria you often need to get into the details of how composability and interoperability is enabled to ensure it meets your functional goals. Just because software “has an API”, **does not mean it will be granular or performant enough, or cover all your application management needs**.

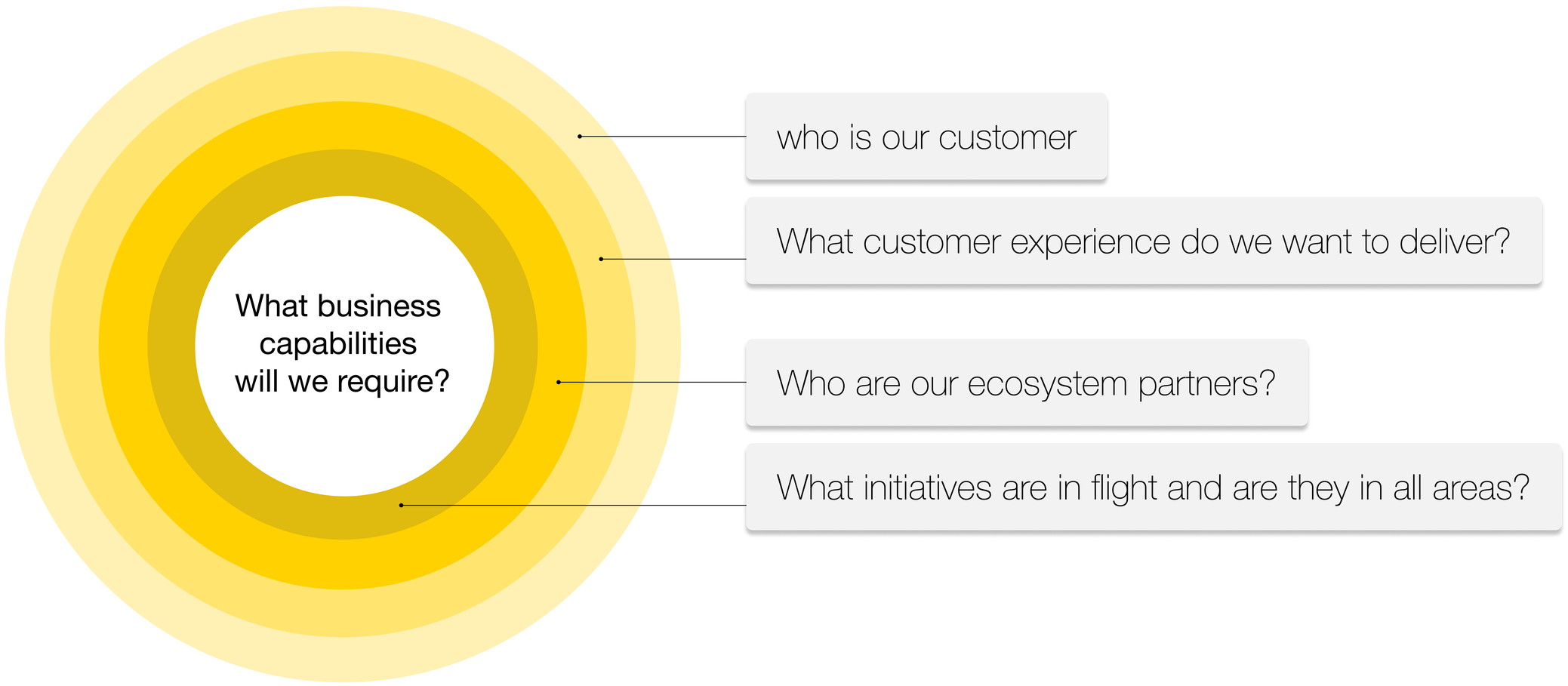
That said, the majority of organizations have to work with legacy data and systems, but you don’t need to let that stop you from transitioning to a MACH-based approach. Rather than a traditional approach of “big bang” migrations which are time-consuming and often prone to failure, a staged approach with carving off smaller business functions will encourage a cycle of bigger wins and learnings over time.

# Appendix

## Time Analysis

### Intermediary layers

Understand the vision of the business. Who are our customers? How do we want them to experience our platform? Who are our current ecosystem partners and what business capabilities do we want to provide?




### Align the vision

With the current and future architecture in mind, what are the headlines of a strategic architecture. What are the metrics that best describe the current architecture and what would be the main beacons of a target architecture.

### Summary of the strategy

Summarize your strategy in a concise statement that captures the essence of the re-architecture goals

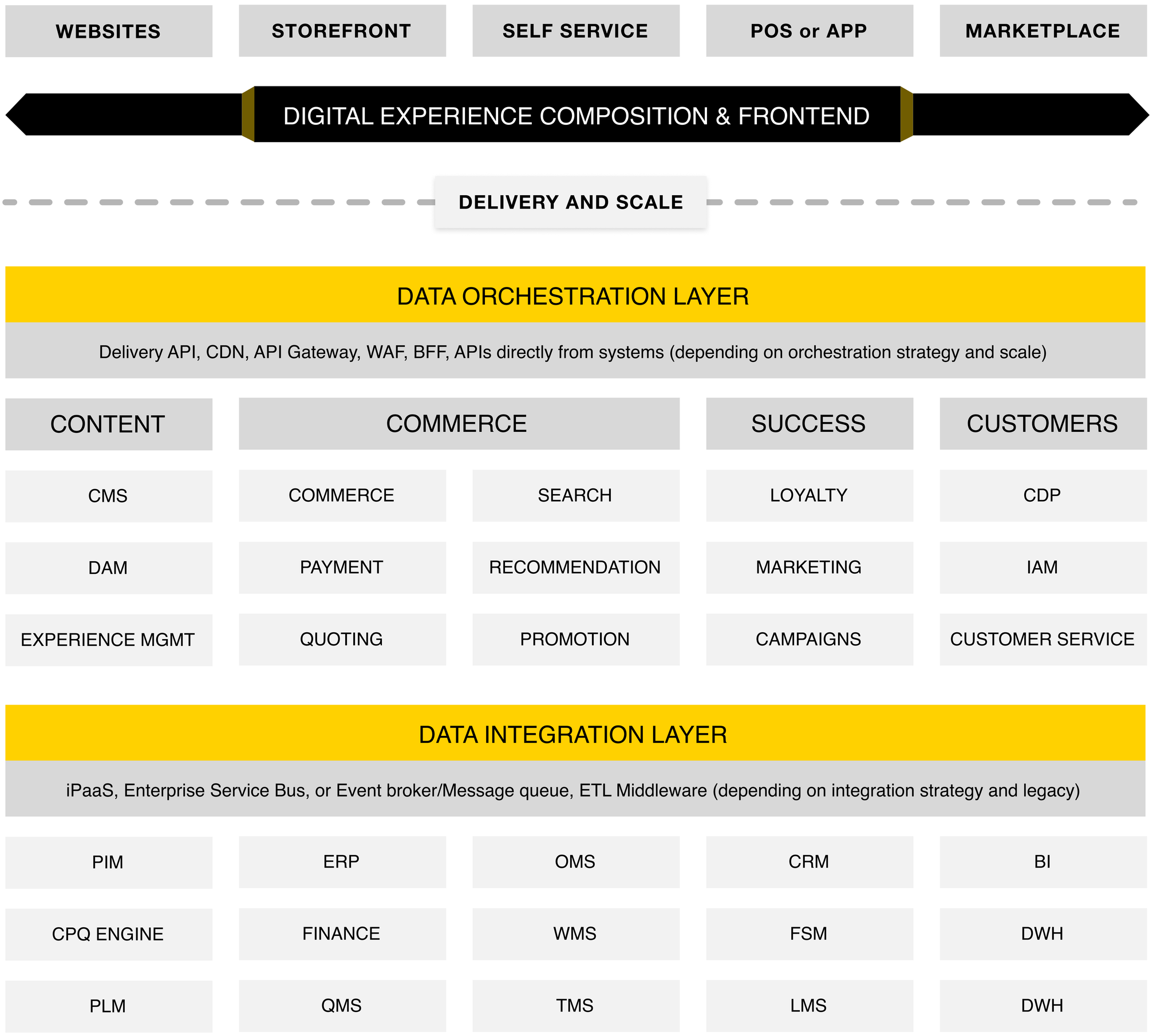
| Top 5-7 Metrics Describing the Initial State | | Target objectives | |
|---|---|---|---|
| We have a reliance on internally built software | Our content takes x weeks to be translated | Enable business composability, speed up time to market | What objectives would show the target architecture adoption has been successful? |

# Appendix

## Time Analysis

### Identify and classify the components in the estate

An enterprise is made up of a variety of components that all work in unison to allow the business to operate. Identify all the components in the system and classify them into various macro domains.



# Appendix

## Time Analysis

### Identify and classify the components in the estate

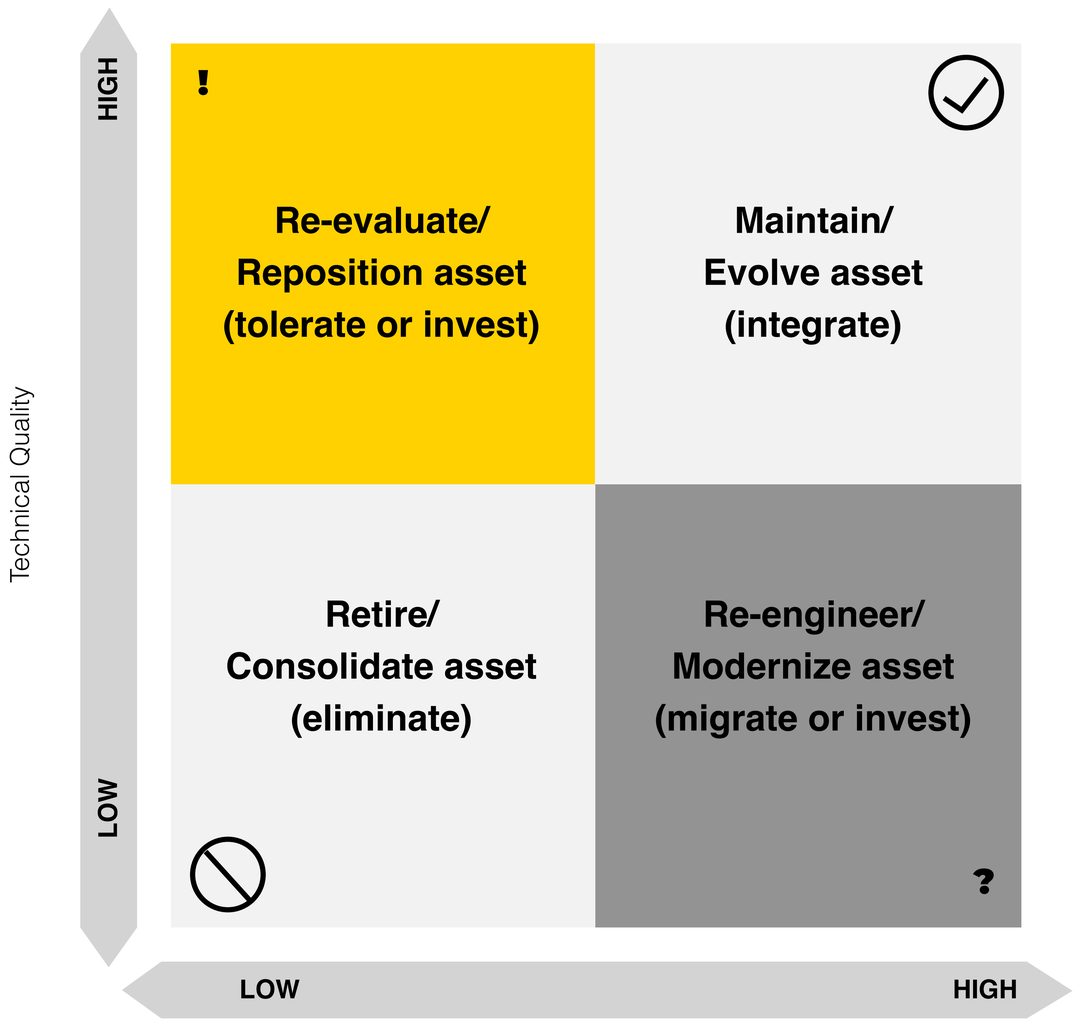
An enterprise is made up of a variety of components that all work in unison to allow the business to operate. Identify all the components in the system and classify them into various macro domains.

**Tolerate** — the application is in good technical shape but lacking in business support, so IT would tolerate it in the portfolio until such time as the business wanted to invest in improving its business fitness.

**Invest** — the application is in good shape, so you should invest in it when asked to add features or turn on some new functionality of a packaged application, while also keeping it technically healthy.

**Migrate (or Modernize)** — The application does just what the business wants, but IT is concerned with the age and brittleness of the underlying technology. If the business wants functional improvements, IT should try to address this technical debt simultaneously by migrating the technical stack or the packaged application to current, supported technology.

**Eliminate (or Replace)** — These applications may be in such bad shape it is not worth spending on them. If they are not needed, or the functionality is now available in a better application, they could be eliminated. If the functionality is still needed, they might need to be replaced.




## Considerations

**Application value and fit to the mission, mandate or process needed**

- Mission or process fit
- Data and information quality/timeliness
- Application robustness
- Utilization
- Future role of the organization

**Operational risk**

- Complexity
- Reliance on subject matter experts
- Maintenance change factors
- Supportability
- Availability and cost of support skills

**Technical risk**

- Architectural alignment or compatibility
- Base technology quality
- Extendibility and scalability
- Technical execution

**Costs**

- License and support contract
- Service maintenance and enhancements
- Total application life cycle costs

MACH ALLIANCE

Stay updated with the latest news and content from the MACH Alliance.
Sign up for our newsletter here


e. info@machalliance.org
w. www.machalliance.org