Addressing legacy constraints

Building flexibility into your technology platforms to enable business agility

MOZAIC

The future of the Operating Model

> As a recognised leader in IT and Digital Operating model design and transformation, Mozaic has delivered wholescale change in over a hundred, large complex estates over the past 10 years – possibly more than any other single organisation during that period. Our team includes ex-CIOs and CTOs from across a broad range of industries, giving us a unique perspective on the past, and on the next phase of operating model change that will affect us all.

THE SERIES

This whitepaper is one of a series that looks at the future of the operating model and details the specific areas of change that organisations will need to embark upon to transform to Enterprise Product and achieve excellence in technology delivery.

- The future of the technology operating model
- Focusing on value
- The importance of culture in transformation
- Measure the things that really matter
- Aligning sourcing models to support Enterprise Product
- Value stream management - it's time to stop throttling change
- Data driven operations
- Addressing legacy constraints

The full catalogue of papers can be found on the Mozaic website at https://mozaic.net/insights/.

Accompanying the series, Mozaic offers a range of complementary workshops, which look in more detail at the subject areas, and help teams to better understand the challenges and opportunities in their context.

If you would like to know more, please contact us at info@mozaic.net or call us on 0203 709 1625.

Enabling agility

> The latest evolution of the Technology Operating Model, Enterprise Product, offers significant advantages delivering greatly improved technology and business agility. But speed to market is often constrained by legacy platforms, which are complex, time-consuming, and expensive to change. In this whitepaper, we discuss how these constraints can be addressed by aligning responsibility into the right area of the Operating Model and iteratively decoupling elements of the platform through the implementation of microservices architectures.

Enterprise Product is an approach to aligning business and technology with the services that offer value to customers and users, rather than functional departments or processes. This approach has several advantages, including:

- **Improved customer focus**: enabling teams to stay closely aligned with their customer's needs and preferences, making it easier to develop products and services that meet these needs.
- **Improved collaboration:** By breaking down functional silos and organising teams around products, the model encourages greater collaboration and teamwork, which leads to more creative solutions and better outcomes.
- **Increased innovation:** Teams are empowered to experiment to develop and refine products and services.
- **Improved accountability:** Teams are accountable for product outcomes - a product-centric model promotes a culture of ownership and responsibility, which can lead to better quality products and services.
- **Faster time-to-market:** By streamlining processes and focusing on value, a product-centric model helps organisations get new products to market more quickly and efficiently.
- **Greater agility:** The model enables organisations to respond more quickly to changing market conditions, customer demands, and competitive pressures.

Of course, speed to market and agility are dependent upon the technology platforms that are continually maintained and enhanced.

A legacy system is one that constrains the business when it needs to change. In this context, a poorly architected system can be regarded as legacy on its day of launch. Old, monolithic systems based on dated technology don't stand a chance.

Constrained by your legacy?

Large, monolithic, highly integrated systems exist in most organisations, and these are usually accompanied by fear and loathing - they're seemingly too big and scary to replace, and a constant source of frustration to customers, business users and IT alike.

Maintaining a legacy platform can be a challenging and resource-intensive task. They invariably use outdated technology, have limited integration, lack documentation, and many suffer from highly coupled architectures that lead to scalability issues. It is not uncommon for a legacy platform that was built initially as a back-office system, to subsequently become a critical part of a high-volume, high-availability online platform.

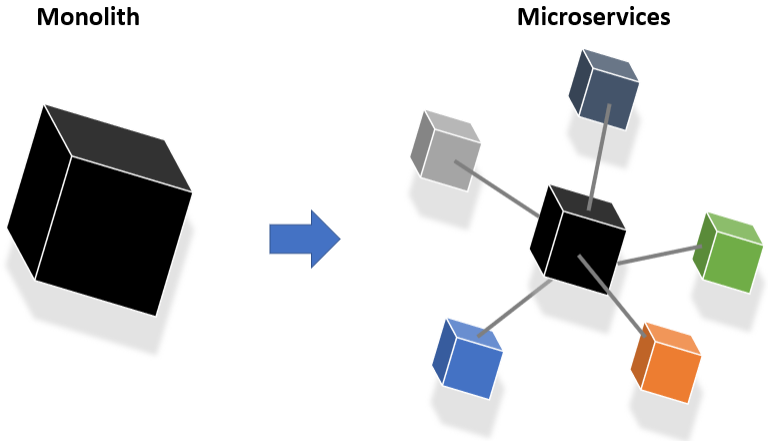
Legacy platforms are in essence a specific, and particularly unpleasant, case of technical debt. The complex, monolithic nature not only makes it difficult to make changes but to do so often introduces further complexity. This leads to high levels of technical debt, which makes maintenance more difficult and expensive. It's a vicious circle.

Maintaining a legacy platform can be a huge challenge and a significant constraint, requiring expertise and investment to overcome the issues associated with outdated technology.

However, a proven solution to this problem now exists. **Breaking these complex systems into smaller independent applications and services, using a microservices approach, enables change to be made in small, safe, incremental steps.** Taking this approach means that each change can be delivered quickly whilst enabling parallel working on multiple business needs.

The advantages of Microservices

Microservices are a well-established, proven, software architectural paradigm that involves breaking down large applications into small, independent services that can be developed, deployed, and scaled independently. Each microservice focuses on a specific task or function and communicates with other microservices through well-defined APIs.



Microservices provide a proven approach through which the challenges of legacy platforms can be addressed. Importantly they can be implemented incrementally, reducing the constraint, and adding flexibility in a prioritised order. There is no requirement for a massively complex and expensive “big bang” change.

Scalability Microservices enable you to scale different parts of your application independently, allowing you to respond to changes in demand effectively.	**Flexibility** With microservices, you can develop and deploy each service independently, which makes it easier to make changes to your application.	**Better fault isolation** Microservices allow you to isolate and identify faults more easily, which makes it easier to fix issues independently
Maintainability Microservices are designed to be modular, making it easier to test, maintain, and update individual services without affecting the entire system.	**Faster time-to-market** With microservices, you can develop and deploy individual services more quickly, which can help you get new features to market faster.	**Technology diversity** Different technologies can be used for each service, making it possible to use the best tool for the job, including the latest innovations.

Overall, a microservices architecture offers a more flexible, scalable, and resilient approach to application development that can help you respond to changing business needs more effectively. Importantly, it provides the flexibility to enable the technology agility required by product teams.

Organising to address legacy debt

Addressing legacy debt is not simply a technology issue. **To succeed, it is essential that the change approach is baked into the operating model, and that resources and processes are aligned in the resolution of debt.**

In general terms, teams should be delegated the responsibility to address debt as part of their day-to-day activity of delivering value. And value is key – debt should be addressed (ideally through the implementation of microservices) only where its resolution delivers clear value to the business. Debt resolution should not be an end itself and should not lead to big-bang legacy replacement. Unless, of course, there is a significant value driver e.g. the replacement of a truly obsolescent system.

As such, debt resolution should be included in the appropriate teams' backlogs, agreed with Product Owners and SMEs, and prioritised accordingly. Importantly, their development should be communicated widely to ensure appropriate sharing and reuse.

PRODUCT OR COMPONENT SERVICE

Typically, microservices are developed and deployed at the component level, in that they are underpinning services that can be shared by different lines of business, departments and Products. Sharing of component services allows for their efficient reuse across the organisation.

However, there are cases where the microservices themselves can be viewed as a type of product and therefore have dedicated product teams.

A good example of this is the development of microservices in the retail banking sector to meet the PSD2 (Open Banking) policy requirement. Each service had clear value in that it met a regulatory requirement - it was therefore a requirement to do business. Hence, the majority, if not all banks, built Open Banking product teams to develop, own and maintain the microservices.

Although the new services were initially intended simply to provide APIs to the legacy platform, in many cases they became more functionally rich, decoupling functionality from the core and providing clear value to the customer.

By design these services provide clean access to the legacy platforms and new functionality, thus addressing the legacy constraint for other product teams, and thereby becoming the preferred method for core banking integration i.e. it's much quicker to access the legacy platform through Open Banking APIs than it is to commission new services. As such, these services morphed into both products and component services.

Although this is a specific case, it is likely that many organisations will encounter similar opportunities and organise accordingly to focus on value.

Migrating to Microservices

Migrating from a legacy system to a microservices architecture can be a complex process, but there are some general steps you can follow to make the transition smoother:

1. **Understand your legacy system:** Before you begin the migration process, it's important to understand your legacy system thoroughly. Document its architecture, components, and dependencies, and identify areas of the system that may be particularly challenging to migrate.

2. **Define your microservices architecture:** This involves identifying the services that will make up your new system and defining how they will communicate with each other. The architecture should be decoupled with clean APIs.

   This should include the ownership between product teams and how they will work together where there are cross-dependencies.

3. **Identify the services to migrate first:** Break down the functionality of your legacy system into smaller, more manageable services. Agree on a prioritisation strategy – our recommendation is provided in the next section.

4. **Develop and test the microservices:** Incrementally develop and test each microservice individually. This will help you identify any issues early on and ensure that each service works as expected. It also ensures a naturally progressive return on investment.

5. **Migrate data:** Migrating data can be a complex process, especially if you're moving from a monolithic database to a distributed data architecture. Develop a plan for migrating data and test it thoroughly to ensure that data integrity is maintained.

6. **Monitor and maintain the microservices:** Once your microservices are deployed, it's important to monitor them closely and address any issues that arise. Use tools like logs and metrics to identify issues and optimize your microservices over time.

Remember that the migration process will likely take time and require significant resources. Be prepared to invest in the migration process and be patient as you work through any challenges that arise.

Other considerations

PRIORITISING YOUR APPROACH

When embarking on your Microservices journey, it is important to agree on your implementation strategy. Although this may evolve over time, it is useful to guide teams and help with integration planning. Mozaic's proposed approach is as follows:

- Identify and decouple simple "edge" components.
- Minimise the dependencies of any decoupled component with the monolith.
- Use dependency and structural code analysis tools to identify the most coupling and constraining factor capabilities in the monolith, and then decoupling those areas.
- Decouple capabilities vertically from the core capability with its data and redirect all front-end applications to the new APIs. In this way, we avoid the anti-pattern of only decoupling facades, only decoupling the backend service and never decoupling data.
- Prioritise the decoupling of what is most important to the business and changes frequently.
- Decouple capability and not code.
- Decouple capabilities with appropriate boundaries, not going too small.
- Continue to decouple in a continual approach, with a series of migration steps.

STANDARDISATION AND PATTERNS

One danger, when implementing your architecture, is that insular product squads focused on their own customers' challenges can lead to the introduction of duplication, a lack of consistency, and waste. Here, the solution is the development, management and reuse of standard patterns and components. This is particularly relevant in the development of Microservices that will span the enterprise.

The de-coupling of business functionality should naturally divide between product teams, whereas service such as Application Login may lead to a "Common Component" team, which may not only be centralised but develop these to be used across many applications.

Ensuring that reusable patterns are collated and made readily available to each Product team is fundamental to sustaining the velocity of change. A full library of components also significantly improves the quality and predictability of delivery as well as enabling enterprise-wide controls, security, and management effectiveness.

To succeed, each pattern needs ownership within the IT/Digital organisation as a technical product or service component; capability assigned for its ongoing support and improvement; a backlog of change which should be managed and prioritised against product team needs; to be stored within a visible an easily accessible library that is available to all product teams; and to be built into the orchestrated flow of work as default capability.

What are you waiting for?

Business success is now inexorably linked to the success of IT delivery, and product-oriented operating models are proven to support agility. Enterprise Product builds upon these models to drive greater value.

As we have seen, business agility cannot be achieved if legacy platforms are allowed to constrain technology delivery. Microservices architectures provide a proven approach to addressing this challenge, an approach all organisations should look to take.

A full architectural transformation is not a simple task; microservice requires significant investment and ongoing support and maintenance. But the investment is worth making.

Importantly, as microservices are small, decoupled objects, they can be developed and delivered incrementally, releasing value steadily over a long period, reducing risk and proving the value return.

Technical debt resolution should not be an end in itself, but often organisations find that the implementation of Open APIs provides new opportunities to monetise existing data and functionality. In the case of Open Banking, there are examples of retail banks now offering value-add services through third parties that integrate with their legacy platforms.

If you'd like to know more about the approach or are embarking on your journey and would like to benefit from our deep experience, please contact info@mozaic.net, or contact either of the authors – contact details on the following page.

[END]

FOR MORE INFORMATION

STEVE TUPPEN
07584 171 013
steve.tuppen@mozaic.net

RUSSELL SMITH
07980 624 738
russell.smith@mozaic.net

LEGAL DISCLAIMER