OPEN INNOVATION

www.openlegacy.com

# Legacy Modernization in the World of APIs, DevOps and Microservices

Discover the pros and cons of 'modern' approaches to legacy migration to enable innovation and speed

Craig Beattie and Patty Hines, CTP

*"Your legacy IT infrastructure likely isn't going anywhere soon and may not actually be the problem. You need to* ***bridge your legacy*** *back-ends with DevOps to meet business requirements, and microservices is a way to do it."*

Zeev Avidan, CPO, OpenLegacy

CELENT

# LEGACY MODERNIZATION IN THE WORLD OF APIs, DEVOPS AND MICROSERVICES

Craig Beattie and Patricia Hines, CTP
07 December 2018

OLIVER WYMAN

## CONTENTS

## EXECUTIVE SUMMARY

### KEY RESEARCH QUESTIONS

**1** *What does a modern architecture look like, and how does it enable innovation?*

**2** *What is a legacy system in this modern architecture?*

**3** *Is building a new digital company from scratch the only way to be a truly modern financial institution?*

Much of the focus on digital transformation in financial services is on customer-facing, external channels, when in fact, substantive transformation opportunities lie across front, middle, and back offices. To serve fast-moving customers with changing expectations, financial institutions must achieve a new level of agility and automation.

The combination of microservices, APIs, and DevOps allow financial services firms to transform their legacy architecture, isolating the limitations of these critical systems. APIs also create a bridge between traditional batch-based, on-premises integration approaches and real-time digital integration with the cloud, mobile, and social applications underpinning omnichannel delivery.

Over the past 10 to 20 years, while systems integration technologies that harness service-oriented architecture (SOA) and web services have improved greatly, core business operation applications and business operation processes have remained largely unchanged. Monolithic legacy systems continue to hamper financial institutions' efforts to include value-added services for customers, posing challenges in terms of agility and innovation.

Leading banking and insurance organizations are using a combination of microservices, DevOps, and APIs to move to a modern, digital architecture using more flexible and agile development and deployment methods.

# THE DIGITAL SHIFT AND INNOVATION

Radical changes in technology, coupled with rapid consumer adoption, and rising staff expectations around usability have brought significant shifts in how financial services are delivered and implemented.

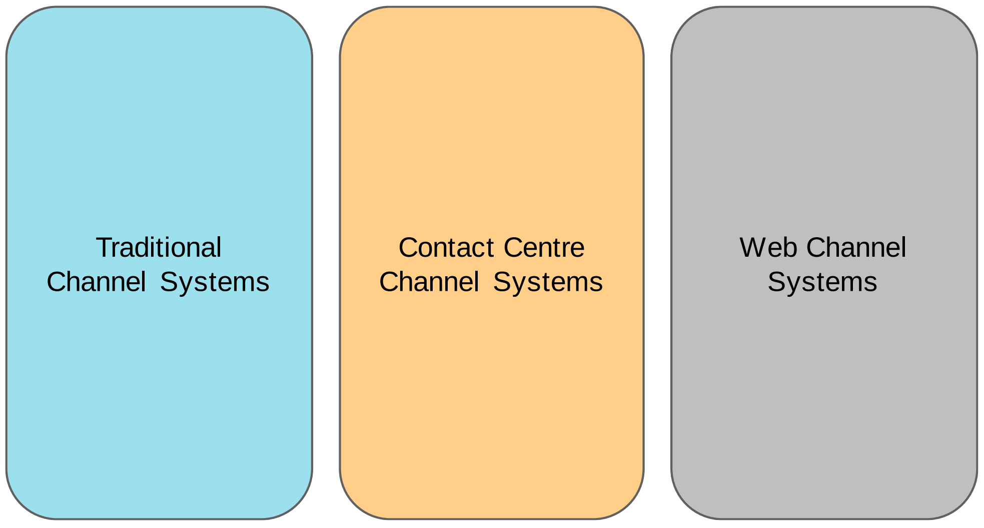
## The Move from Physical to Virtual Institutions

Most associate the start of digital with the rise of the Web. However, the rise of the Web followed the earlier shift from bricks and mortar to the telephone, where expensive branches and high street presence gave way to financial services delivered through call centres at lower operational costs. Many significant brands grew in the 1980s out of this shift and form the large incumbents still operating today.

The rise of the Web and the move online shifted the world to a 24/7 model, where self-service and e-commerce became king. Many legacy systems and legacy business models were nearer to 12/6 than 24/7, and this shift led to a rise in new entrants touting 24/7 support. Even the new batch of contact centre institutions had to scramble to accommodate these new requirements.

In response, many incumbent financial institutions built adjacent architectures next to their existing systems — systems that supported the existing products and sales approach but weren't 24/7.

Figure 1: The Rise of the Channel Silos



Source: Celent analysis

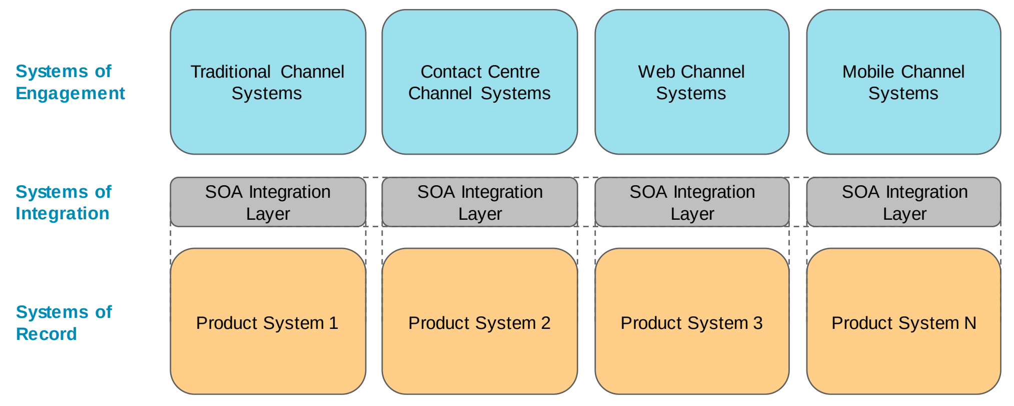
## Going Omnichannel and Early APIs

After the rise of the Web and now smartphones the idea of a web-only or phone-only institution was reserved for the niche play. Larger financial institutions and those that prided themselves on customer service found that they needed to support incoming requests in one channel — say the Web, but then successfully complete them in a branch or on the phone. The concept of adding a new stack of technology for each new channel was not only cost-prohibitive but also wouldn't satisfy customer needs.

A maturing occurred, and SOA was seen as a means to expose services from the existing systems that the various channels could then consume. The goal here was to successfully support multiple channels and have consistency across those channels.

The early SOA APIs were a means to an end often generated directly from the source, systems as shown in Figure 2.

Figure 2: The Use of an SOA Layer



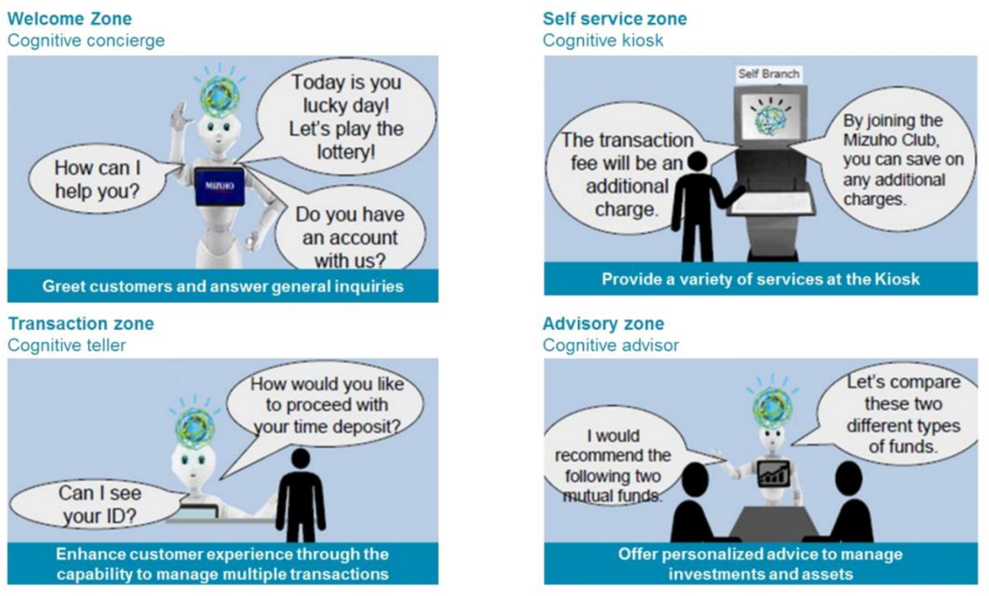
Source: Celent

In Figure 2, we see that in this pattern SOA services are built in front of the systems of record to enable the systems of engagement. In this case the integration layer uses just enough of what is available to enable as much as possible in the channels.

## Financial Institutions Are Enabling Ever More Complex Channels

With the rise in the types and complexity of new channels, APIs are critical to cost-effective adoption. Below are two examples from Mizuho Bank in Japan.

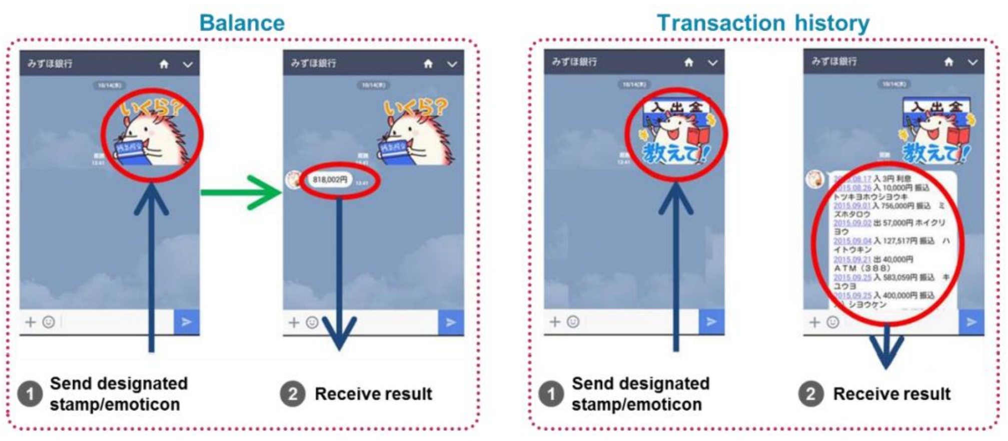
Figure 3: Illustration of Pepper Use Cases


Welcome Zone
Cognitive concierge
Today is you lucky day! Let's play the lottery!
How can I help you?
Do you have an account with us?
Greet customers and answer general inquiries
Self service zone
Cognitive kiosk
Self Branch
The transaction fee will be an additional charge.
By joining the Mizuho Club, you can save on any additional charges.
Provide a variety of services at the Kiosk
Transaction zone
Cognitive teller
How would you like to proceed with your time deposit?
Can I see your ID?
Enhance customer experience through the capability to manage multiple transactions
Advisory zone
Cognitive advisor
I would recommend the following two mutual funds.
Let's compare these two different types of funds.
Offer personalized advice to manage investments and assets

Source: Celent, *Mizuho Bank: AI and Social Media Banking*, April 2017

While Mizuho Bank's use of emojis is unusual, the rise of AI-powered chatbots throughout the retail industry and in financial services has been extraordinary.

Figure 4: Sample Chatbot

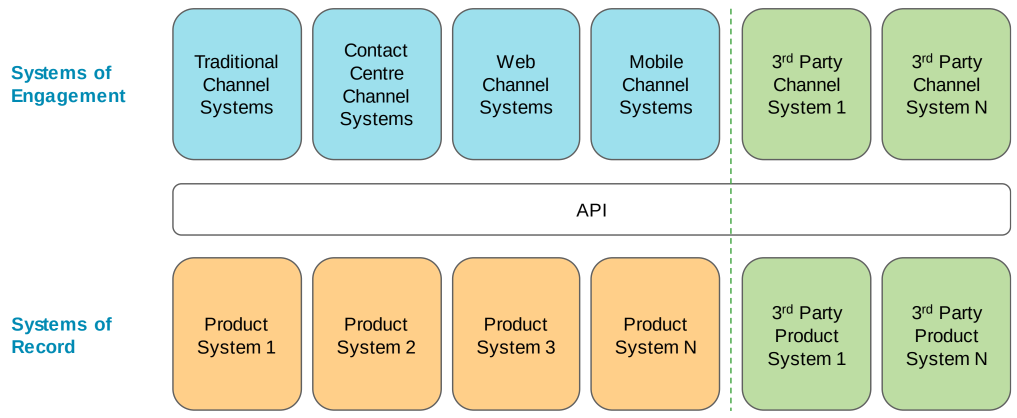

Balance
Transaction history
1 Send designated stamp/emoticon
2 Receive result
1 Send designated stamp/emoticon
2 Receive result

Source: Celent, *Mizuho Bank: AI and Social Media Banking*, April 2017

## API-First

The current shift observes that the quality of the API layer is the primary driver of value. Far from being a byproduct of creating a modern financial institution, the API is the asset that allows for innovation and for participation in a wider ecosystem, and reduces both cost and delivery timescales. With an API-first design philosophy the API itself is given focus, and is the asset that is shared beyond the enterprise.

Figure 5: Logical Architecture Showing API Layer


Systems of Engagement
Traditional Channel Systems
Contact Centre Channel Systems
Web Channel Systems
Mobile Channel Systems
3rd Party Channel System 1
3rd Party Channel System N
API
Systems of Record
Product System 1
Product System 2
Product System 3
Product System N
3rd Party Product System 1
3rd Party Product System N

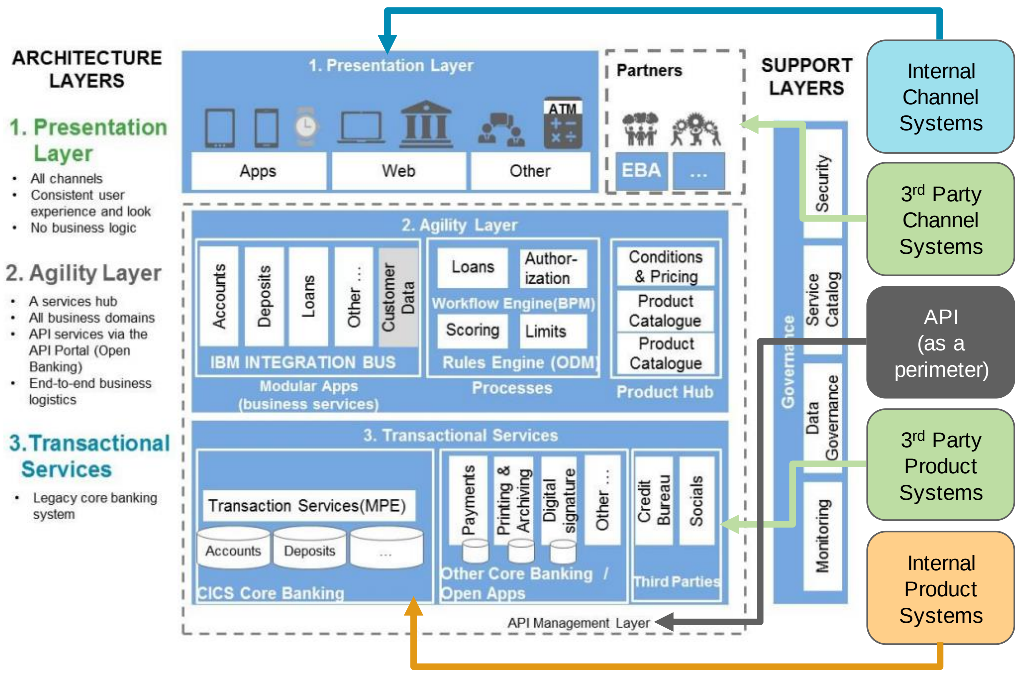
Source: Celent analysis

In Figure 5, the API is expressed as a single layer. In some enterprises, the public API is offered as a single layer through a single API Gateway. In a microservices architecture this API may be implemented as many discrete applications, perhaps operating as a microservices mesh without the need for a centralised implementation.

Regardless of how it is implemented or realised, a well-designed API allows both internal systems and third party organisations to leverage services from within the financial institution. Further, the financial institution can offer products from partners and weave the offering into their own channels. The opportunities this allows are discussed further in the next section.

A sample concrete implementation of the architecture is offered in Figure 6, where we see Intesa San Paolo weaving partners into their architecture at the front end and third parties offering transactional services at the back end.

Figure 6: Solution Overview at Intesa San Paolo (Pre-Microservices)



Source: Celent, *2018 Model Bank Winner for Modernising IT Architecture: Intesa San paolo*, April 2018

## Enabling Innovation

While some believe that fintechs will eventually make banks obsolete, the current reality is that banks and fintech companies are entering collaborative partnerships for innovation, giving banks access to innovative products, new technologies, and startup culture, and giving fintech companies access to funding, regulatory expertise, and customer reach. We continue to see press announcements of banks enabling fintechs, and fintechs enabling banks, as shown in Figure 7. This trend is also prevalent in the insurance industry and their relationships with insurtech firms.

Figure 7: API Partnerships Between Banks and Fintech Firms Provide Business Value to Both

**Bank – Fintech API Partnerships**

| | | |
|---|---|---|
| citi | QANTAS | In 2017, Citi deployed 70 APIs to build a Qantas branded credit card and mobile application in less than 9 months. |
| fidor BANK | bitcoin.de | Fidor enables bitcoin.de, a crypto-currency trading service, to display Fidor customers on its trading portal, identifying accounts with real-time clearing. |
| CBW BANK | Moven | FDIC-insured CBW helped Moven to launch its digital only bank in 2011, supplying its platform and leveraging its banking license. |

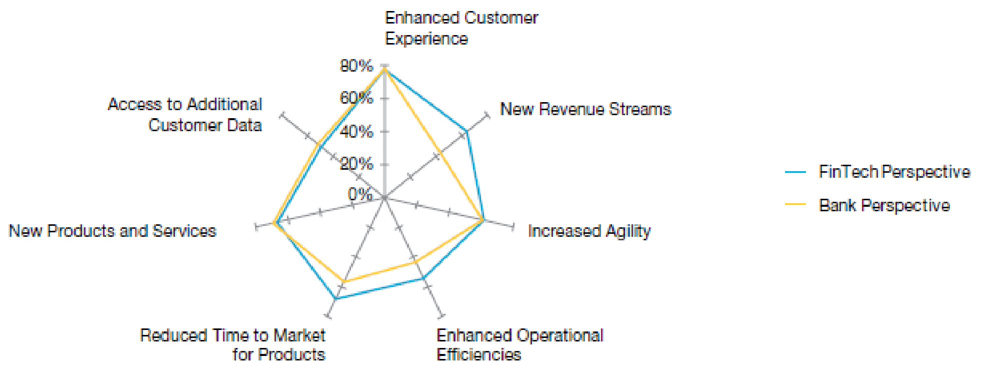
**Fintech – Bank API Partnerships**

| | | |
|---|---|---|
| LevelUp | CHASE | Enables Chase Pay users to use the LevelUp app to order ahead and pay at participating quick-service restaurants; gives access to special offers. |
| Moven | TD | TD MySpend is a companion app to the TD mobile app for both Android and iOS platforms, leveraging Moven's personal financial management tools. |
| Bill.com | Capital One | Enables Capital One small business customers to leverage the Bill.com platform as a method to pay bills, invoice customers, and collect payments. |
| DWOLLA | BBVA Compass | Enables BBVA Compass account holders to make real-time payments to each other at a low cost. |

Source: Company announcements and websites

Many of these API partnerships depend on customer data and services embedded in legacy technology. By working together and taking advantage of APIs, banks and fintech firms, insurers and insurtech firms leverage their distinct strengths, enhancing the customer experience much more than each entity could do on its own.

Figure 8: Benefits of Implementing APIs for Banks (%), 2017

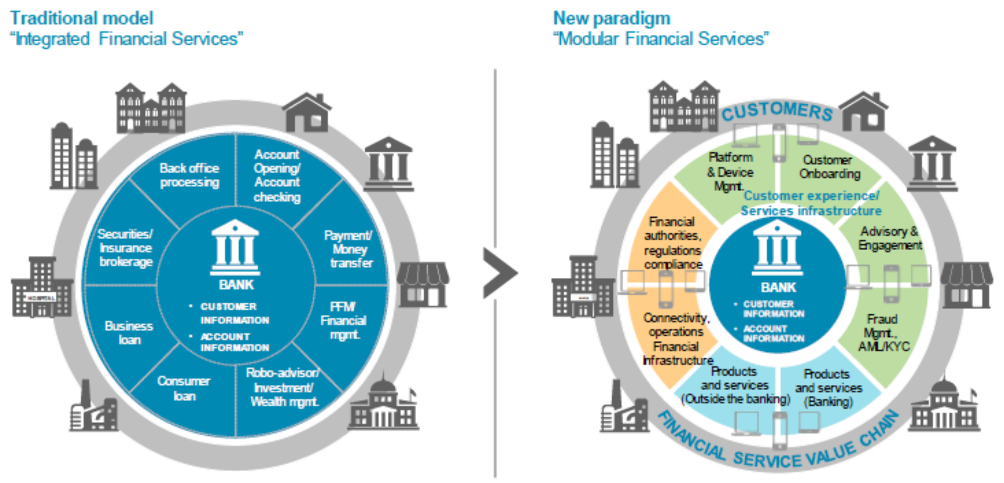


Source: Capgemini Financial Services Analysis, 2017; 2017 Retail Banking Executive Interview Survey; Capgemini Global Financial Services

## Modular Financial Services

In harnessing microservices, DevOps, and Open APIs, financial institutions need to undertake a paradigm shift to transform themselves into digital financial services companies. Microservices, DevOps, and Open APIs are not ends in and of themselves but rather a means to modularization. In harnessing these technologies and development approach, financial institutions can undertake a paradigm shift to transform themselves into providers of modular financial services, rather than monolithic, integrated financial services.

Figure 9: The Future of Financial Services: Modular Financial Services



Traditional model
"Integrated Financial Services"
Back office processing
Account Opening/ Account checking
Securities/ Insurance brokerage
Payment/ Money transfer
BANK
• CUSTOMER INFORMATION
• ACCOUNT INFORMATION
Business loan
PFM/ Financial mgmt.
Consumer loan
Robo-advisor/ Investment/ Wealth mgmt.
New paradigm
"Modular Financial Services"
CUSTOMERS
Platform & Device Mgmt.
Customer Onboarding
Customer experience/ Services infrastructure
Financial authorities, regulations compliance
Advisory & Engagement
BANK
• CUSTOMER INFORMATION
• ACCOUNT INFORMATION
Connectivity, operations Financial Infrastructure
Fraud Mgmt., AML/KYC
Products and services (Outside the banking)
Products and services (Banking)
FINANCIAL SERVICE VALUE CHAIN

Source: *Microservices: A Software Engineering Revolution Beyond the Clouds*, Celent

***Key Research Question***

**1**

*What does a modern architecture look like, and how does it enable innovation?*

Increasingly a modern or digital architecture features APIs and microservices, and leverages significant automation as seen in DevOps.

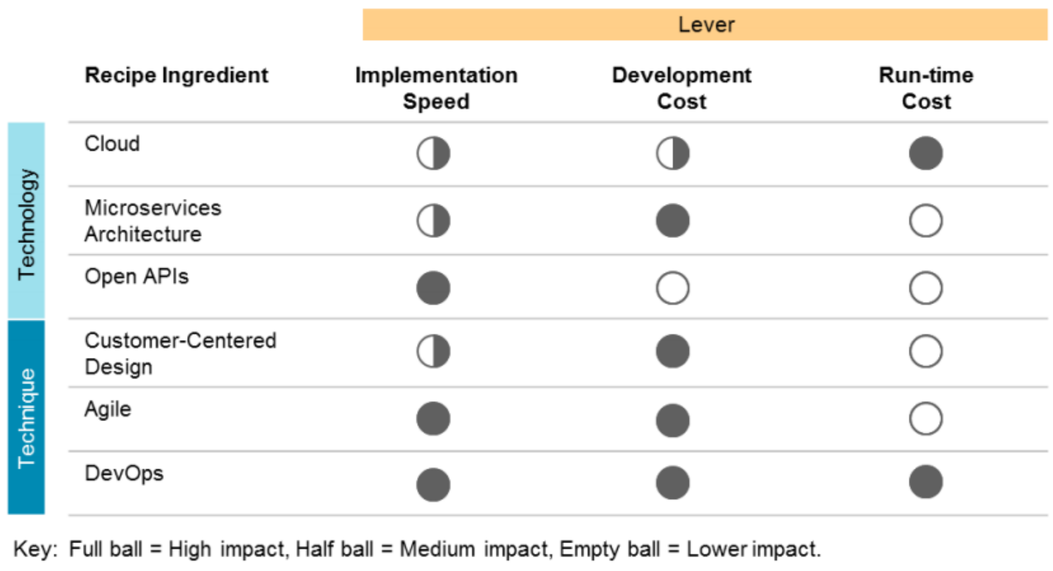
# TRANSFORMING LEGACY SYSTEMS

## THE LEGACY CHALLENGE

Historically, when people in financial services talked about legacy, images of mainframes running 40-year-old applications came to mind — in fact many of the old guard in the IT departments of financial services firms still remember punch cards as a means of entering programs into computers.

Figure 10 discusses the ingredients in this new recipe for IT and the relative benefits compared to a classic IT approach.

Figure 10: Benefits of the New Recipe



| | | Lever | | |
|---|---|---|---|---|
| | **Recipe Ingredient** | **Implementation Speed** | **Development Cost** | **Run-time Cost** |
| Technology | Cloud | ◐ | ◐ | ● |
| | Microservices Architecture | ◐ | ● | ○ |
| | Open APIs | ● | ○ | ○ |
| Technique | Customer-Centered Design | ◐ | ● | ○ |
| | Agile | ● | ● | ○ |
| | DevOps | ● | ● | ● |

Key: Full ball = High impact, Half ball = Medium impact, Empty ball = Lower impact.

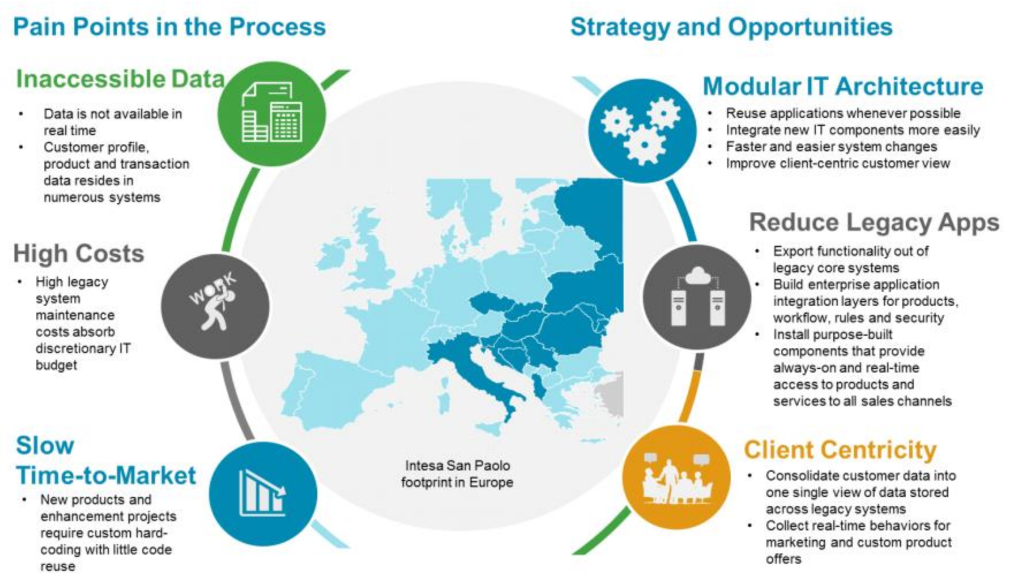
Source: Celent Report: *The New Recipe That Is Changing Insurance*, February 2018

In this context, legacy systems are perceived to disable financial institutions in terms of agility, speed to market, and flexibility in how they distribute their products because the new definition of digital technology is simply much faster.

In short, legacy systems didn't get slower. Instead, the bar has been raised in terms of speed of change, quality, and cost.

The challenges facing Intesa San Paolo and the opportunities are neatly expressed in Figure 11.

Figure 11: Summary of Intesa San Paolo's Challenges and Objectives


Pain Points in the Process
Strategy and Opportunities
Intesa San Paolo footprint in Europe
WORK

**Inaccessible Data**

- Data is not available in real time
- Customer profile, product and transaction data resides in numerous systems

**High Costs**

- High legacy system maintenance costs absorb discretionary IT budget

**Slow Time-to-Market**

- New products and enhancement projects require custom hard-coding with little code reuse

**Modular IT Architecture**

- Reuse applications whenever possible
- Integrate new IT components more easily
- Faster and easier system changes
- Improve client-centric customer view

**Reduce Legacy Apps**

- Export functionality out of legacy core systems
- Build enterprise application integration layers for products, workflow, rules and security
- Install purpose-built components that provide always-on and real-time access to products and services to all sales channels

**Client Centricity**

- Consolidate customer data into one single view of data stored across legacy systems
- Collect real-time behaviors for marketing and custom product offers

Source: Celent, *2018 Model Bank Winner for Modernising IT Architecture: Intesa Sanpaolo*, April 2018

In the Celent report, *2018 Model Bank Winner for Core Banking Transformation: Zions Bank*, April 2018, Zions Bank expressed the challenge thus:

- **Costs**: Maintaining legacy systems carries a higher cost than running modern core banking systems due to the number of workarounds. Integration work is expensive, because there are many more risks of opening up a system that is built to be left alone.
- **Flexibility**: Large mainframe-based legacy platforms were built for stability and speed and to process millions of transactions in a batch at the end of the day. They weren't made to be altered. New and changing banking functionality requires ongoing development. It's very difficult, and will only become more difficult, to develop modern, flexible customer experience on top of legacy cores.
- **Developer talent**: Many of the developers who originally worked on some of these core platforms are at the end of their careers (or lives). New IT pros are attracted to more modern architecture written in languages like Java or C#. Younger developers often have a hard time making sense of the vast interconnectedness of banking systems, many of which are poorly documented. This hampers their ability to deliver projects and injects substantial risk because pulling on one string can unravel another. Switching to a modern core (with real-time accounting and componentized architecture) is seen as job enrichment to the IT talent coming onstream today.
- **Ability to serve a digital customer**: Tech companies are leading the way with what's possible in digital. Customers expect a modern digital experience, and legacy core systems have been challenged to deliver it. Core systems that lack capabilities around real-time, cloud readiness, componentization, and openness today find it more difficult and expensive to keep pace with the competition; and the task will only become more challenging.

The results in the six months after National Australia Bank's modernisation are a great example of the increase in agility and speed to market that can be achieved.

Figure 12: National Australia Bank's Results

| METRIC | VALUE | CHANGE |
|---|---|---|
| NEW ACCOUNTS PER MONTH | 20,000 | N/A |
| TIME TO FUNDING: CREDIT CARD | 2.6 days | 50% |
| TIME TO FUNDING: PERSONAL LOAN | 2.4 days | 59% |
| INCREASE IN APPROVAL CONVERSION RATE | N/A | 20% |

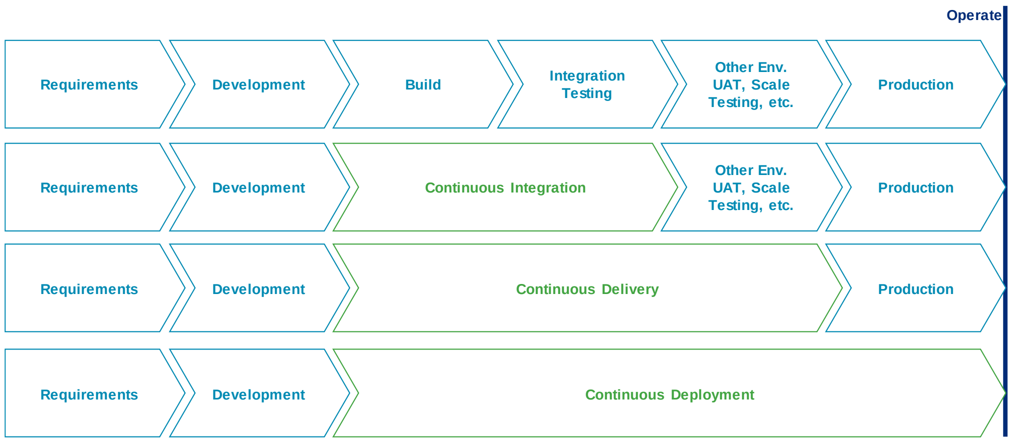
Source: Celent, *National Australia Bank: Personal Banking Origination Platform*, April 2017

## Waterfall Vs. Agile Vs. DevOps

Historically IT change projects and programs were frequently delivered in a waterfall style reminiscent of large engineering deliveries. Focus was on defining up-front the requirements and then designing for the entire deliverable. This would then be built and go through progressively more intense testing phases to prove what was built was what was requested.

Agile methods and now DevOps seek to reduce the time between request, test, and delivery, to deliver in multiple, shorter deliveries and to automate as much of this process as possible. Figure 13 shows how continuous integration helps automate build and integration testing activities, and then how continuous delivery and then deployment increase the amount of automation.

Figure 13: Evolution of Automation in Development



Source: Celent Report, *Building Your DevOps Chops: A New IT Approach Aimed at Faster/Better/Cheaper*, October 2016

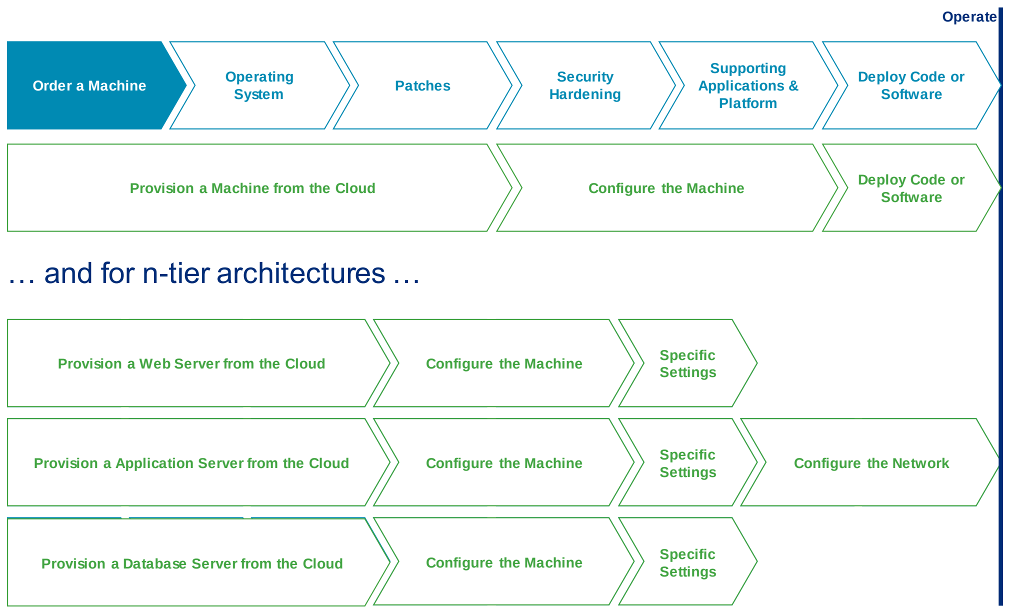
## On-Premise Monoliths Vs. Microservices in the Cloud

The Ops side of DevOps provides even greater possibilities for increased speed of IT change. Historically in deploying a solution one would have to order the machines, set up the operating system on them, patch them, and harden them for security, and that was before putting the application on and doing business.

When commissioning infrastructure is so costly, one makes sure it is highly utilised. Further one doesn't commission infrastructure needlessly. Thus, even web-scale architectures ran on a limited number of machines.

Today, a machine can be requested from the cloud, automatically configured, and even linked to other machines in the network in minutes, rather than months as shown in Figure 14.

Figure 14: Provisioning and Configuring Machines Now Takes Minutes



Source: Celent Report, *Building Your DevOps Chops: A New IT Approach Aimed at Faster/Better/Cheaper*, October 2016

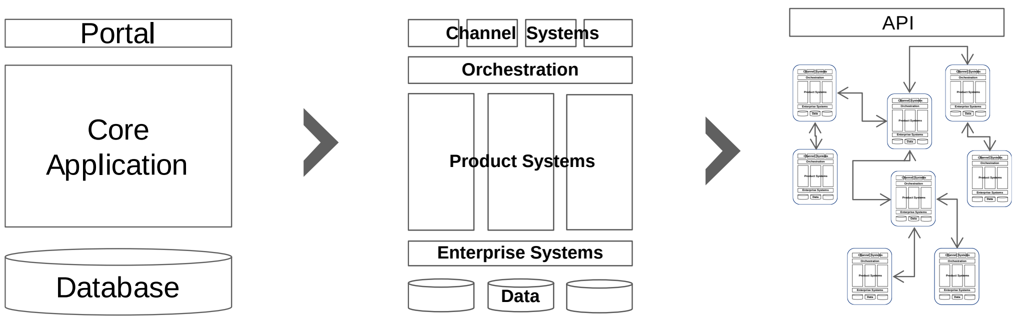
DevOps and Agile are not new to the IT industry, but parts of the financial services industry have been slow to pick them up. To give some real context to the benefits one might expect from these approaches, here is what the early adopter Hiscox reported in *Celent Model Insurer 2015: Case Studies of Effective Technology Use in Insurance*:

- Delivering the project on time and under budget, which is a first for Hiscox with projects of this scale.
- The platform has enabled Hiscox to move from its typical 10-week waterfall delivery model to an iterative, two-week cycle.
- Minimising the number of defects in the first few weeks (post go-live). The number was well below Hiscox’s expectations.
- Delivering fixes and product configuration changes into production faster. Thanks to the tools introduced through DevOps, a release takes six minutes (on average) vs. three hours with the previous technology stack. In the week prior to go-live, Hiscox performed 47 releases, which would have been impossible with the traditional deployment tools.
- Since launch, Hiscox has deployed 26 times per week (on average) across the nine environments. The average cost savings of each automated deploy has equated to approximately £7k a week. This equates to a savings of approximately £370k annually.
- Additional benefits of consistency, time saved investigating manual errors, and time saved training new staff has resulted in cost avoidance. At its peak, the team performed 19 deployments in one day!

In a world where an application can be deployed to its own infrastructure in minutes, software architects are allowed to reconsider the size of a component and how it can be deployed independently of the rest of the infrastructure. Further, the servers become disposable — if a server starts to misbehave, then destroy it and start a new one. There’s no need to “rescue” a server that may have been hacked or has a bug.

In this environment, an API is delivered by many independent applications, running on their own servers, being automatically tested, automatically maintained, and automatically scaled as necessary.

Figure 15: Evolution of Software as Automation Increases

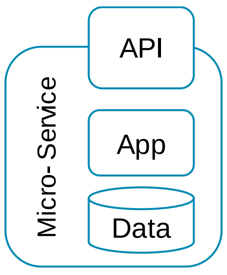


Source: Celent Report, *Building Your DevOps Chops: A New IT Approach Aimed at Faster/Better/Cheaper*, October 2016

## Creating and Deploying a Microservice

Microservices are a popular choice for modernizing legacy systems. Microservices architectures focus on delivering small, discrete, and individually deployable services — in some cases, each service is its own microapplication with an API, some logic or application code, and data.

Figure 16: The High-Level Structure of a Microservice



Source: Celent

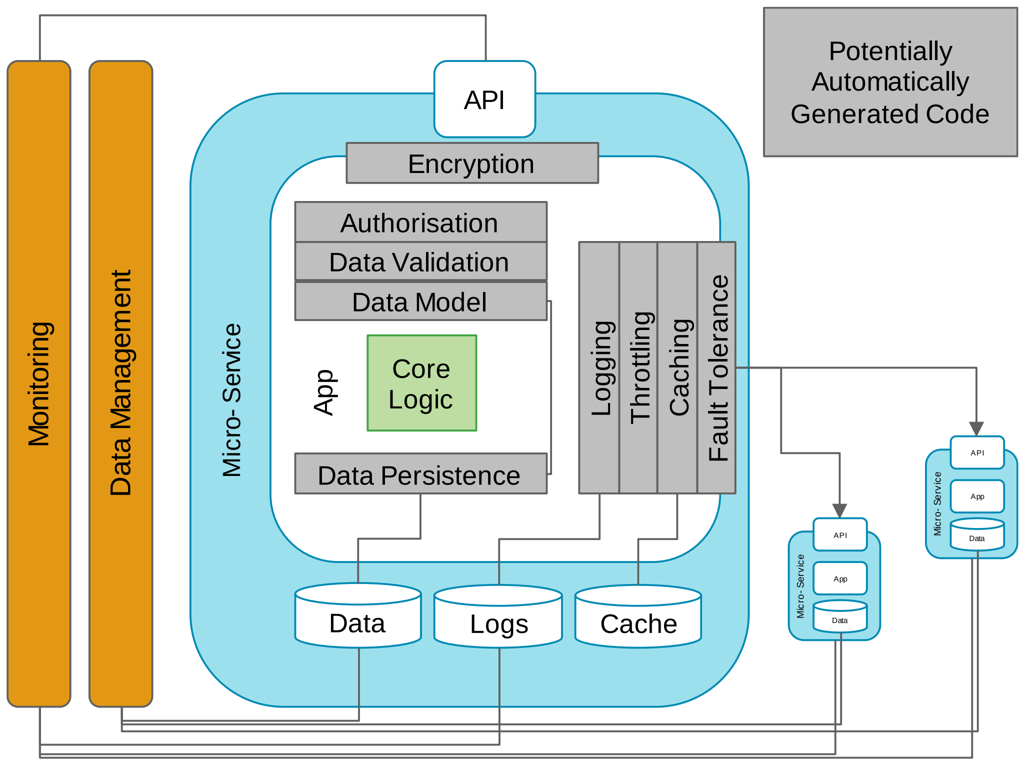
Thus, the name derives from being a collection of many small services. With discrete services, similar, multiskilled teams using accelerated toolsets may support a set of microservices driving an API. As detailed in the Celent report *Honey, I Shrunk the Services: Microservices in Insurance*, (December 2017), the presence of an API does not require a microservices architecture. However, typical microservices architectures focus on delivering an API, aggregated together into a wider API.

Having small components that can be swiftly deployed to machines in an elastic infrastructure allows for a highly scalable and adaptive infrastructure. Microservices provide common business capabilities, accessible through an API, such as “Retrieve Customer Name,” “Create Internal Transfer,” and “Request Credit Line Increase.” A microservices-led approach enables reuse of services and thus reduces integration cost and complexity.

### The Anatomy of a Microservice

When we look deeper into a microservice and see the complexity therein, building many microservices can be overwhelming until we observe how much of that complexity can now be automated.

Figure 17: Inside a Microservice



Source: Celent

By consistently using automation and tools to build microservices it is possible to standardize and introduce best practices, such as securing each service, using logging to monitor and maintain each service and using caching where necessary for performance, fault tolerance, or simply to protect vulnerable legacy systems.

Microservices architectures only work through automating the majority of the microservices build effort.

## The Legacy Trade-Off

One answer to legacy modernization is to get rid of all the legacy systems and replace them with highly automated, fast to build microservices. While this is a laudable goal, most legacy systems represent many tens of thousands of man-hours of effort, and they embody complex rules and often have complex data in them. Replacing them and building all the functionality they offer is a nontrivial task.

Most legacy systems have some strong features in their favour:

- They work
  Legacy systems often support the current business and processes.
- They're stable
  Legacy systems have been around long enough that many bugs have been fixed. With age comes stability.
- They're cheap
  Frequently legacy systems have no or a nominal license fee. Further the infrastructure they run on is often well understood, optimised, and run at low cost.

Replacing legacy systems for the sake of closing them, where there is no business case to doing so, can be a bad investment.

While there is great advantage to moving to an API-oriented microservices architecture, the simple fact is most software in financial institutions is older and simply wasn't built that way. These systems, while limiting in some ways, are still valuable. The financial services industry needs a pragmatic approach to move their technology, where it makes sense to do so, and leverage their legacy assets where it does not.

Below we look at some approaches financial institutions and software vendors to financial institutions are taking to modernise their infrastructure.

***Key Research Question***

**2**

*What is a legacy system in this modern architecture?*

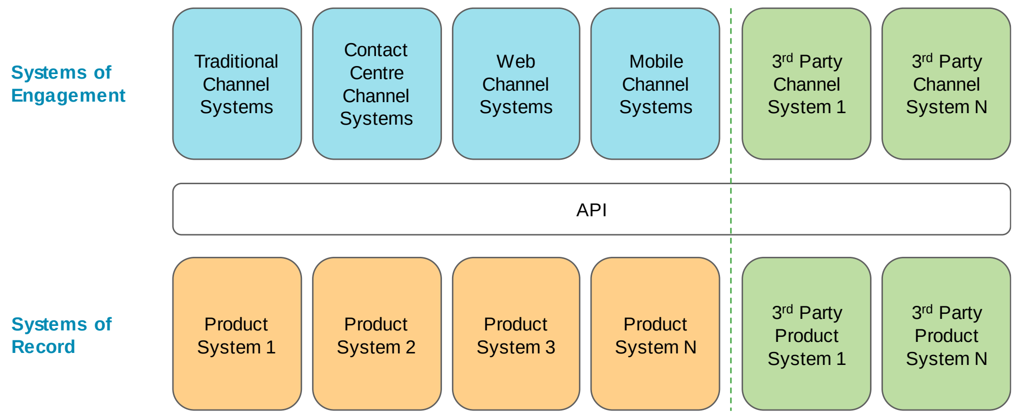
Legacy was perceived to be certain technologies such as COBOL or assembler, or the age of systems, those 20 years old. Today, the defining feature of legacy systems is that they are too slow to change.

Legacy systems are those systems that disable the enterprise, regardless of technology or age.

## A NEW LEGACY MODERNISATION

If we revisit Figure 5 from the first section, copied here as Figure 18, then this pattern offers a different perspective on this logical architecture.

Figure 18: Financial Institutions and Vendors to the FS Industry Are Extending Their Functionality

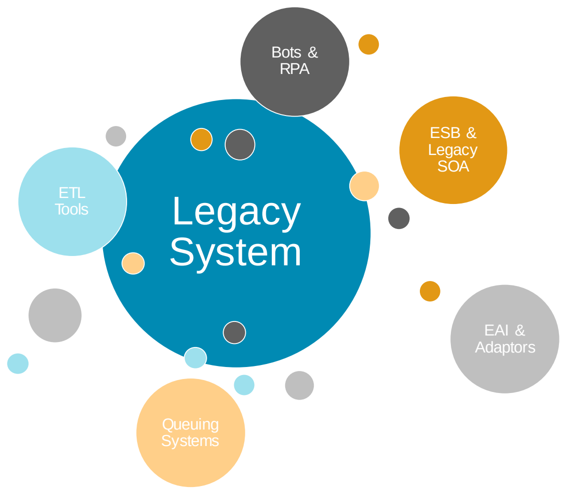


Systems of Engagement
Traditional Channel Systems
Contact Centre Channel Systems
Web Channel Systems
Mobile Channel Systems
3rd Party Channel System 1
3rd Party Channel System N
API
Systems of Record
Product System 1
Product System 2
Product System 3
Product System N
3rd Party Product System 1
3rd Party Product System N

Source: Celent

If one has multiple systems of record that are fit for purpose, then it makes sense to not migrate them. If we take the legacy definition above, if product systems 1 through N are not disabling us, if we can build an API in front of them, then it is reasonable to continue to use them. Most legacy systems have a surrounding set of systems to support them or to act as workarounds; see Figure 19.

Figure 19: The Systems Supporting Legacy Contribute to Their Technical Debt

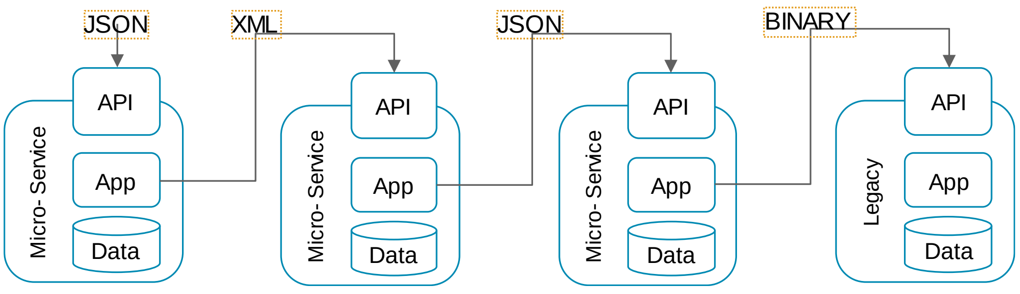


Source: Celent

These surrounding systems can be costly to run and maintain and are absolutely a target for improving efficiency as the influence of the legacy system is reduced. While those systems might initially enable the phased migration approach, over time these surrounding systems could be switched off. Financial institutions have reported reducing API build times in an environment like Figure 19 from weeks to minutes, when supported by automated tooling.

To dig into the operational cost of this technical debt, simply analyse the number of transformations the data goes through as it passes through the architecture. In Figure 20 we show a heterogeneous set of microservices calling through to a legacy SOA component and ultimately a COBOL program. The table below show's each time the information is transformed, including the encryption and decryption steps.

Figure 20: The Challenge of Many Layers and Too Many Transformations



| | | | |
|---|---|---|---|
| Decrypt | Decrypt | Decrypt | Decrypt |
| JSON -> JavaScript | XML -> Python | JSON -> Java | Binary -> Cob |
| JavaScript -> XML | Python -> JSON | Java -> Binary | |
| Encrypt | Encrypt | Encrypt | |

Source: Celent

While the use of a microservices naturally leans toward a highly communicative architecture, one must be aware of the costs of unnecessarily deep architectures leveraging many calls. Pragmatism is needed as well as the courage to address middleware systems that no longer add value.

## A Pragmatic Path to Digital

The great challenge is to hide the implementation of the API from the surrounding applications. If this can be realised it greatly eases the modernisation from the legacy system to the new.

Historically, early SOA attempts with Enterprise Application Integration (EAI) suites and later with Enterprise Service Buses (ESB) often allowed the generation of connectors to the legacy systems, but these proto-APIs often surfaced the problems with the underlying systems straight into the API. The API-First concept is a direct response to this and places design of the API above the implementation.

Modern digital teams then will design the API, then may generate the bulk of the microservices code to meet the desired API. Where legacy system access is required to realise or execute the API, then the system owner may still use automated tools to generate code that allows access to the legacy system. In some cases this may be created as a microservice.

As a result, some mapping between the internal API and the desired API may be needed; depending on the complexity this might be configured or may be coded as another microservice.

Many microservices and API management platforms focus on newer programming languages, applications, and use cases. Using one of these platforms to access legacy system data requires that developers understand the underlying data structure and programming language (often COBOL) and manually code the microservice and accompanying API. Recognizing the challenges, and importance, of modernizing integration to legacy systems, several vendors offer enabling solutions for mainframe connectivity; see Table 1.

Table 1: Vendors Offering API Solutions for Mainframe Applications (Not Exhaustive)

| VENDOR | PRODUCT NAME | DESCRIPTION |
|---|---|---|
| IBM | z/OS Connect Enterprise Edition | Single, focused REST API entry to all Z Systems subsystems. Integrated REST API editor enables design and mapping. API discovery via dynamically created Swagger documents. |
| MULESOFT | Catalyst Accelerator for Banking | A set of API designs and supporting reference implementations that accelerate the path toward digital transformation. Provides a microservices foundation for implementing Open Banking and PSD2 use cases. |
| OPENLEGACY | Microservice-based API Integration & Management | Easily and automatically create microservices-based APIs from legacy systems, including back-end mainframes, midrange systems, applications and databases, and stored procedures as a self-contained standard Java component. Speed delivery of digital transformation projects to days or weeks, simplify complex architectures, and improve performance. |
| ROCKET SOFTWARE | Rocket API | Enable real-time access to critical business functions from virtually any application at a fraction of the time, expense, or risk normally associated with modernization projects. |

Source: Company websites

Success with microservices and API enablers for legacy systems depends heavily on the vendor's approach to automation, standardization, testing, and legacy system connectors. Ideally, the solution should automatically analyze the underlying application to create modern code that developers can choose to transform into web services, microservices, or REST APIs. Prebuilt connectors and templates (e.g., CICS or DB2) help developers to use those web services, microservices, and APIs to build new solutions or enhance existing ones.

## Mitigating Legacy

Legacy is not defined by age of a technology, or a particular type of technology; rather it is a simpler, more binary judgment:

**Does the technology enable the financial institution to achieve its goals and objectives?**

The goals of financial services are being rewritten by experiences and changes outside the industry. Other industries are radically advancing expectations in terms of speed, agility, and the human experience for staff, partners, and customers alike. The goal posts have shifted, and the old technology, the old methods, the use of developers to hand-code — all of this is now too slow, too legacy.

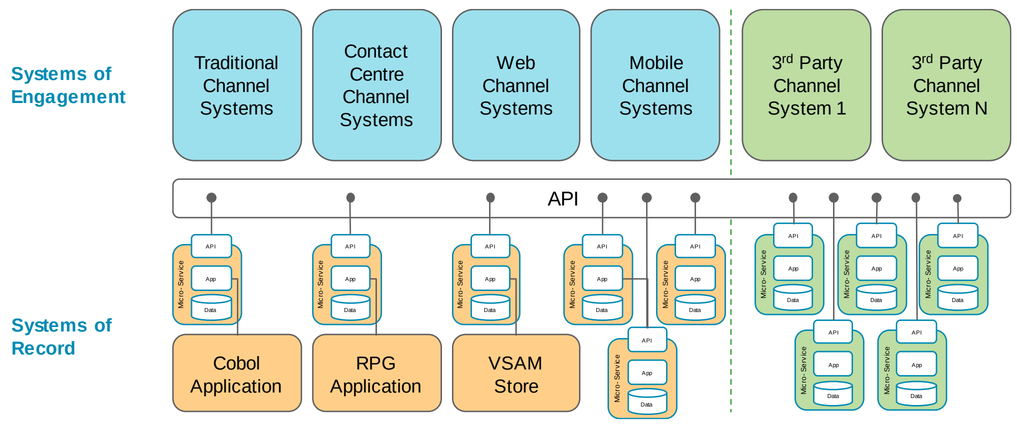
However, as we have seen above, financial institutions are adapting, are adopting new methods, and have already paved the new routes to legacy modernisation. This vanguard has been able to modernise where needed by adopting new techniques, technologies, and opportunities, but also to rediscover the agility and speed in the legacy systems.

A pragmatic approach to mitigating and limiting the impacts of legacy leads us to re-enable the legacy systems. Where possible, if the legacy systems can be super-charged, can become quick enough, and hit the new bar, then they can play a role in helping an organisation meet its objectives.

## Legacy Systems as Participants in a Microservices Architecture

For the foreseeable future many incumbent organisations will target an architecture like that shown in Figure 21, where multiple legacy systems coexist with microservices to deliver an API consumed by multiple clients.

Figure 21: Mixed Legacy and Microservice Architecture



Source: Celent

By leveraging an API as the facade in front of the implementations we can move to a world where the API becomes the primary deliverable and we can shift the implementations underneath it. Table 2 offers scenarios where a microservices approach makes delivering new APIs or changes to APIs easier than a traditional approach.

Table 2: Microservices Approach to Change Scenarios

| Change Type | Microservices Approach |
|---|---|
| Removing old microservices and ending support for old features or versions of the API | One great challenge with large code bases is the removal of old or unnecessary code. The key to the challenge is to prove that a piece of code or data is no longer required, and its removal will have no adverse effect.<br><br>Once again, with microservices the challenge is easier. If the API is now satisfied by another application or microservice then this microservice can be switched off and the code archived. Thus, implementing new versions of the API as new services could be beneficial in removing old and unnecessary code in the future.<br><br>Of course, if we are adhering to the encapsulation view of a microservice then some scheme to share data or do a data migration is required to implement this. |

| Change Type | Microservices Approach |
|---|---|
| Addressing legacy implementations or components | A key challenge with large applications or monoliths is, what does one do when the underlying technology becomes insecure, out of date, or simply difficult to support?<br><br>In the monolith scenario, this is the application modernization challenge with various schemes including rewriting the entire application, replatforming the application, or even code conversion where tools are used to map the system from the current state to a new target one. These all have their challenges.<br><br>In a microservices architecture each service is an application in its own right. This means that each service can be modernized independently of the others — in fact there is no requirement for a common architecture among components (beyond the obvious such as supporting them at a reasonable cost). Thus, if a service that implements an API needs to be modernized or replatformed then the whole, small application can be replaced and reimplemented in the new target method. For instance, if a tool was used to generate some of the early applications and this falls out of support, or out of fashion, then those components implemented in that way can be rewritten in the latest tool at lower cost than the original implementation. Since much of the test infrastructure exists against the API, this is all reusable with the new component regardless of how it is written.<br><br>**Key Assumption**: It will be faster to write new microservices in the future, with new tools, than it is today. |
| Significant new version of an API | In a monolith architecture with one application, this would require a development team to check out part of the application, or mark part of the application for change. It is unlikely in an active application that this is the only change to continue, so these changes must be made at the same time other development teams are making their changes. Each concurrent change is thought of as a branch. The changes from all teams are then merged together. The merged code is referred to as the trunk — continuing the tree analogy. The great challenge in these applications is, as the number of branches or concurrent changes increase, the merging process is higher complexity and higher risk. This phenomenon is often referred to as “trunk clash.”<br><br>In a microservices architecture the APIs that are being changed will be implemented by one or more applications. This allows not just one, but possibly multiple teams to work on the changes in unison, avoiding challenges such as trunk clash and allowing for existing functions unrelated to this change to be left completely alone. |
| Support for a new API feature | Significant new features in this architecture typically mean delivering an extension to the API. This allows clients to adopt the new feature as they need to.<br><br>With a microservices-based architecture a new feature can be delivered as a discrete application. The team can deliver the new functionality independently of the rest of the code base and the work of the other teams, avoiding the issues surrounding “trunk clash,” increasing the productivity of all teams — particularly with large development teams. |
| Introducing a new version of the API to co-exist with the existing version | Let us imagine that a change is required to the API, but there are so many clients using the existing version that we don’t want to force them to change. In effect, the platform must support version 1 and version 2 of the API.<br><br>Microservices offers us the option of having two, independent implementations of the API which both exist in the same architecture but execute as different applications. It is of course possible to have one application which checks for the version requested, but having two allows for an interesting option when ceasing support for old version of the API. |

Source: *Honey, I Shrunk the Services: Microservices and Insurance*, Celent

# THE PATH FORWARD

Legacy isn't slow, it's just not as fast as the digital approach. Not being as fast doesn't mean it is not useful. Legacy systems embody rules, data, and products that are often worth saving, and are costly to reimplement.

While the new startups, fintech and others, favour a digital build all the way through, this is a luxury many large financial institutions cannot afford to impose.

In this report, Celent has shown that there are routes to leverage legacy and be digital, to make the big, slow elephant dance. A pragmatic approach is possible and can benefit from the automation and speed seen from microservices, APIs, and use of DevOps.

***Key Research Question* 3**

*Is building a new digital company from scratch the only way to be a truly modern digital enterprise?*

It is possible to make systems faster and improve time to market with key changes.

Systems previously considered to be legacy can be improved such that they are no longer legacy.

Here are some final tips for those embarking on a pragmatic legacy modernisation strategy:

- **Legacy can stay, automation is non-negotiable.**
  Leveraging modern DevOps practices and fast automation tools is the only way to achieve agility. If legacy methods and tools stay with the legacy systems — they will continue to disable the enterprise.
- **Coordination is the enemy of speed.**
  Having many layers of integration technology and multiple teams to manage them all hurt agility. DevOps sought to automate or streamline design, build, deployment and testing activities —– all historically separate teams. Look at the systems orbiting your legacy for opportunities to do the same. Either these need to be automated, or consider removing them altogether.
- **You can open new revenue streams without addressing legacy first.**
  Celent is observing examples of organisations delivering new products, repackaging old products, and entering new services while living with legacy systems.

The short version of this report might read, "Technology change is getting faster, you need to move swiftly to keep pace." Here we have presented a few routes to take advantage of these technologies — some without losing the value from the legacy systems.

The challenge, of course, is to go and get started.

*Was this report useful to you? Please send any comments, questions, or suggestions for upcoming research topics to info@celent.com.*

# LEVERAGING CELENT'S EXPERTISE

If you found this report valuable, you might consider engaging with Celent for custom analysis and research. Our collective experience and the knowledge we gained while working on this report can help you streamline the creation, refinement, or execution of your strategies.

## SUPPORT FOR FINANCIAL INSTITUTIONS

Typical projects we support related to legacy modernisation include:

**Vendor short listing and selection.** We perform discovery specific to you and your business to better understand your unique needs. We then create and administer a custom RFI to selected vendors to assist you in making rapid and accurate vendor choices.

**Business practice evaluations.** We spend time evaluating your business processes, particularly in legacy modernisation, development tools, and methodology. Based on our knowledge of the market, we identify potential process or technology constraints and provide clear insights that will help you implement industry best practices.

**IT and business strategy creation.** We collect perspectives from your executive team, your front line business and IT staff, and your customers. We then analyze your current position, institutional capabilities, and technology against your goals. If necessary, we help you reformulate your technology and business plans to address short-term and long-term needs.

## SUPPORT FOR VENDORS

We provide services that help you refine your product and service offerings. Examples include:

**Product and service strategy evaluation.** We help you assess your market position in terms of functionality, technology, and services. Our strategy workshops will help you target the right customers and map your offerings to their needs.

**Market messaging and collateral review.** Based on our extensive experience with your potential clients, we assess your marketing and sales materials — including your website and any collateral.

## RELATED CELENT RESEARCH

Mizuho Bank: AI and Social Media Banking
April 2017

2018 Model Bank Winner for Modernising IT Architecture: Intesa Sanpaolo
April 2018

Microservices: A Software Engineering Revolution Beyond the Clouds
April 2018

The New Recipe That Is Changing Insurance
February 2018

2018 Model Bank Winner for Core Banking Transformation: Zions Bank
April 2018

National Australia Bank: Personal Banking Origination Platform
April 2017

Building Your DevOps Chops: A New IT Approach Aimed at Faster/Better/Cheaper
October 2016

Celent Model Insurer 2015: Case Studies of Effective Technology Use in Insurance
March 2015

Model Insurer 2018: Case Studies in Digital and Omnichannel
April 2018

The Year of the Insurance Platform: Property / Casualty Edition
June 2018

Honey, I Shrunk the Services: Microservices and Insurance
December 2017

## Copyright Notice

### Prepared by

Celent, a division of Oliver Wyman, Inc.

For more information please contact info@celent.com or:

| | |
|---|---|
| Craig Beattie | cbeattie@celent.com |
| Patricia Hines | phines@celent.com |

## AMERICAS

### USA

200 Clarendon Street, 12th Floor
Boston, MA 02116

Tel.: +1.617.262.3120
Fax: +1.617.262.3121

### USA

1166 Avenue of the Americas
New York, NY 10036

Tel.: +1.212.541.8100
Fax: +1.212.541.8957

### USA

Four Embarcadero Center, Suite 1100
San Francisco, CA 94111

Tel.: +1.415.743.7900
Fax: +1.415.743.7950

### Brazil

Av. Doutor Chucri Zaidan, 920 – 4º andar
Market Place Tower I
São Paulo SP 04578-903

Tel.: +55.11.5501.1100
Fax: +55.11.5501.1110

## EUROPE

### France

1 Rue Euler
Paris
75008

Tel.: +33.1.45.02.30.00
Fax: +33.1.45.02.30.01

### United Kingdom

55 Baker Street
London W1U 8EW

Tel.: +44.20.7333.8333
Fax: +44.20.7333.8334

### Italy

Galleria San Babila 4B
Milan 20122

Tel.: +39.02.305.771
Fax: +39.02.303.040.44

### Switzerland

Tessinerplatz 5
Zurich 8027

Tel.: +41.44.5533.333

## ASIA

### Japan

The Imperial Hotel Tower, 13th Floor
1-1-1 Uchisaiwai-cho
Chiyoda-ku, Tokyo 100-0011

Tel: +81.3.3500.3023
Fax: +81.3.3500.3059