Scott Logic

The Legacy Modernisation Guide

A step-by-step guide to continual, low-risk legacy modernisation

Contents

We've designed **The Legacy Modernisation Guide** so that you can read it as a step-by-step resource or as individual modules. Use these links to access the module you're interested in:

Drivers of legacy modernisation

Pages 8-10

Mapping your IT estate

Pages 11-13

Analysis and Evaluation

Pages 14-19

Planning and Evaluation

Pages 20-23

Implementation

Pages 24-34

Foreword

We've been helping clients modernise legacy systems since the earliest days of Scott Logic, supporting large-scale transformations that have had a lasting and meaningful impact. Through this experience, we've established and refined our approach to modernisation, distilling a collective knowledge of what good looks like.

The aim of this guide is to share with you that distillation of Scott Logic's knowledge and experience so that you can approach your own legacy modernisation initiatives safely and confidently. While we support the entire modernisation lifecycle, we have focused this guide on the technical and planning challenges you may face, as these are often overlooked.

I'm personally an avid AI enthusiast, and Scott Logic is doing pioneering work to help clients harness AI safely in support of their business goals. Given this, you may be surprised to find few references to AI in this guide. This is intentional.

AI can, and should, assist almost every aspect of modernisation: tools for mapping your technology estate will have AI built in; your AI assistant can help you write your business case; AI coding agents can create test suites for your legacy systems; and AI can transform code bases when you're replatforming.

AI primarily changes the economics. Code was a valuable asset when your legacy systems were written. Now, as AI takes on more and more coding tasks, what stands out is not the ability to write code, but the ability to ask the right questions, design the right solutions, and ensure technology serves its purpose.

That's why there are intentionally few references to AI in this guide. AI won't change the fundamental techniques and approaches to legacy modernisation. It will certainly be a great friend throughout your modernisation journey, but it isn't the complete answer. The stable, tried-and-tested approaches in this guide will steer your path forward.

Colin Eberhardt
CTO, Scott Logic

Your modernisation compass

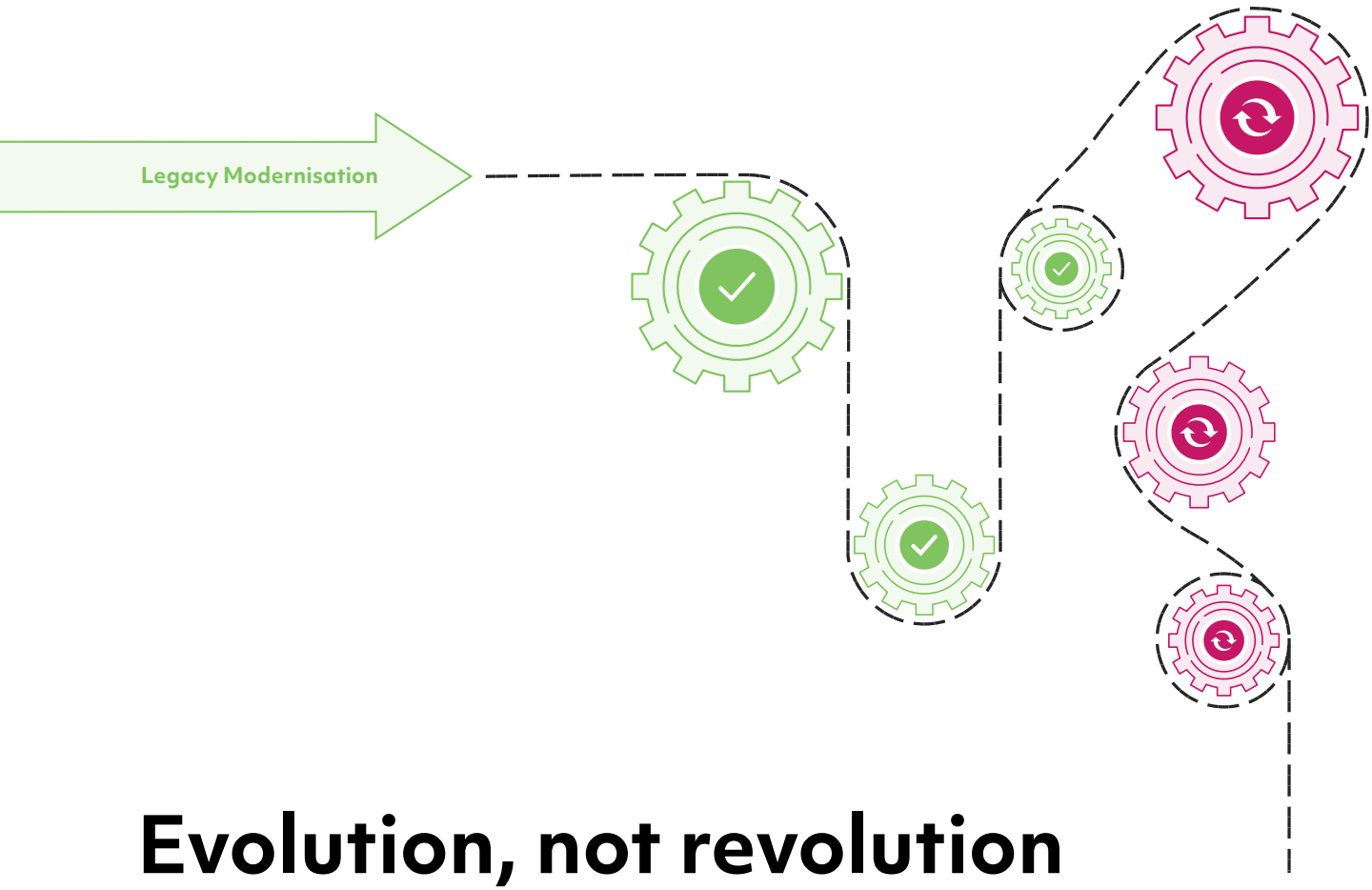
This guide takes you on a step-by-step journey through one cycle of the iterative and incremental approach to legacy modernisation. However, we also provide shortcuts to topics that might be pressing concerns for you as you're reading this.

If you're a Transformation Lead embarking on a legacy modernisation project, there's a range of common objections that you may hear; some may already be familiar to you. We have set them out below, aiming to help you interpret what's actually going on, and the options available to you, with links to navigate you to the relevant part of this guide.

What you've heard	What's probably going on	Options for you to consider
It's too critical a system and too risky to modernise it.	• If it's a core system, they're probably right that it is risky to modernise, but it's better to understand and mitigate the risk now than when your hands are forced by an incident.	• Secure **stakeholder buy-in** by articulating the business risks of doing nothing. • Adopt a risk led **test strategy** up front that validates the highest impact risks first (not just requirements).
We're too busy with BAU to start a migration.	• They're probably seeing migration as a 'Big Bang' effort and not understanding that continuous modernisation can run in parallel with BAU. • They haven't considered that the problem can be decomposed, nor that modernisation could reduce their BAU and legacy technology debt.	• Introduce a **new perspective**: that IT estates are dynamic and modernisation should be a continual process. • Bring in **third-party burst capacity** for short-term projects.
It's too big a problem, we don't know what we've got or where to start.	• They may lack a clear, documented understanding of the IT estate and its system architecture. • They're not thinking incrementally and haven't considered the benefits of mapping the estate organisation-wide. • They've not understood that this mapping can be automated in large part – often, development teams are unaware that operational teams have a view of the estate already using Application Lifecycle Management tools, and the like.	• Understand your organisation's capabilities in mapping its IT estate, then **begin continuous estate-mapping**. • Identify the top modernisation priority and **assess your options** from the **4Rs** and the **spectrum of Rework**.

Continued on next page

What you've heard	What's probably going on	Options for you to consider
If we don't do it in one go, users will be confused.	• Considering anything but a 'Big Bang' approach leads people to assume that user experience will be negatively impacted. • This is particularly the perception when strategies such as Strangler Fig or Vertical Slicing are under consideration.	• Consider UI Modernisation and Desktop Interoperability as the means of decoupling the user experience from work underway on the backend. • Reflect on the potential to streamline business process while modernising.
We'll just end up with one more system.	• Everyone has heard modernisation myths of one system that turned into two and ended up as 13.	• Understand why the Analysis and Evaluation phase is critical, allowing you to decompose the problem and identify the right modernisation strategy/ies. • Understand why sufficient up-front planning will give you a holistic view and enable you to shape an incremental plan. • Have a test strategy that gives you confidence that you've done the right thing and done it well. • Define decommissioning criteria up front and report progress against them openly. Understand that it's important to know when you're done.
It will require lots of licences and hardware	• Teams tend to skip ahead to problems with modernisation approaches. • From their limited experience of coexistence strategies, they have a false impression of the need to procure additional hardware and licences.	• Consider incremental re-platforming and cloud migration.
We don't have the skills in-house to build or maintain a modern system.	• In-house teams necessarily focus on the technology they're supporting, which can prevent them from having a broad view. The learning curve can be steep.	• Introduce a new perspective on legacy modernisation, explaining that continuous modernisation means continuous learning. • Bringing in external help with projects enables experiential knowledge transfer and contextual training.
The system's a monolith and must be replaced as a whole.	• They may lack a clear, documented understanding of the IT estate and its system architecture. • They probably don't know the system well enough to decompose it. • If they do, they probably haven't considered the potential to modernise by making minimal changes without replacing everything all at once.	• Strangler Fig may be an option, or one of the other coexistence strategies. • Rework to existing systems is a viable option through approaches such as API Wrappers or CDC.



Evolution, not revolution

It's fair to say that many organisations used to view legacy modernisation as an eruptive, unforeseen event – a 'Big Bang' – that disrupted business-as-usual and had to be endured.

However, in recent years, technical strategies, approaches and tooling have emerged that have turned legacy modernisation into a source of competitive advantage, rather than a cross to bear.

In the old perspective, IT estates were viewed as essentially static, which rendered organisations passive and reactive in the face of change. Many organisations now understand that IT estates are dynamic and that modernisation must be a continual process underpinned by estate monitoring as a strategic capability.

At Scott Logic, we've seen the benefits to organisations that embrace this approach. Well-serviced systems enable greater operational resilience, meaning fewer outages and higher customer satisfaction. Regular updates to an IT estate promote enhanced security, just as the updates to your phone operating system keep you protected. If your estate can evolve to be more flexible and adaptable, you can keep pace with regulatory changes. The cost of modernisation becomes much more predictable, with smaller, iterative investments, rather than the financial shock of massive one-off projects.

That's why, at Scott Logic, we recommend an iterative, incremental and continuous approach to legacy modernisation. If you're a Transformation Leader or a Technology Leader, this guide will provide you with practical advice on how to adopt this approach. It will offer an overview of this perspective on the legacy modernisation lifecycle and set out the options available at each stage.

Let's begin at the beginning, with the drivers of legacy modernisation.

Drivers of legacy modernisation

Drivers of legacy modernisation

The business strategy of your organisation – and the elements realised through its technology strategy – will be both the driving force and the North Star of your legacy modernisation initiatives. This overarching vision will provide the principles and the roadmap that will guide prioritisation and decision-making.

These architectural principles might include aiming to reduce (or increase) the number of vendors, or to reduce the number of programming languages or frameworks, or to migrate incrementally to the cloud. It's the role of those responsible for enterprise architecture to coordinate efforts across the organisation so that modernisation decisions are aligned to the strategy and its principles. By aligning decisions across teams, architects minimise duplication, avoid fragmented systems, and help the organisation modernise with greater cost discipline and lower risk.

Alongside the strategy and architectural principles, the third key factor influencing modernisation decisions is budgetary. To allocate a finite budget effectively across the IT estate, architects will divide systems into three groups:

Invest

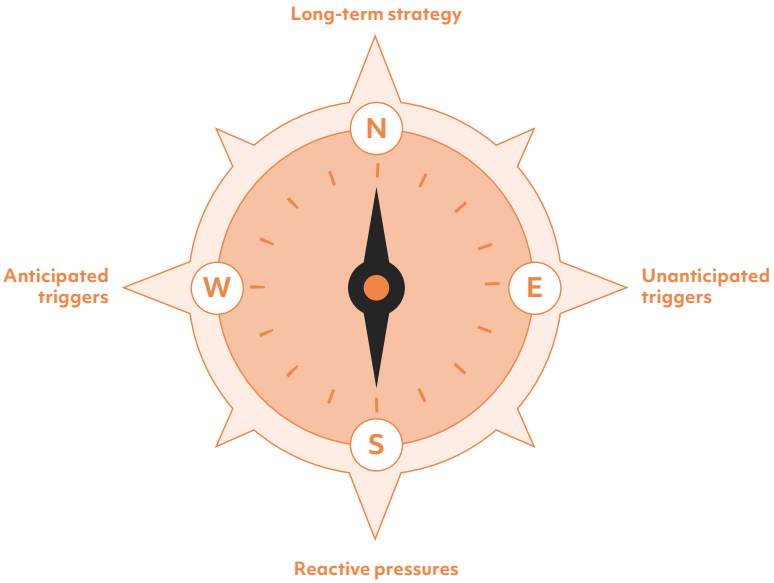
Systems that are strategically important or carry significant risk or inefficiency. These become priorities for modernisation, enhancement, or replacement because investing here unlocks value or reduces major pain points.

Divest

Systems that are no longer worth the cost, complexity or vendor burden. These are candidates for retirement or consolidation, and budget is directed toward activities that help move the organisation away from them.

Maintain

Systems that are stable, meeting needs and not causing issues. These get minimal sustaining investment, freeing budget to focus on the 'invest' or 'divest' candidates.



While many triggers for modernisation are anticipated and reflected in the strategic direction, organisations will always face unplanned pressures that force reactive change. Effective modernisation balances these reactive demands with the long term strategic vision. At Scott Logic, we have supported our clients to strike this balance in the face of a wide range of anticipated and unanticipated triggers; we'll look at some of them below.

Some modernisation triggers

Technological changes

For example, the advent of cloud computing and the rapid advance of Artificial Intelligence (AI) have placed intense pressure on organisations with large legacy estates.

Performance metrics

A system's performance may deteriorate such that it consistently fails to operate within predetermined thresholds.

Data silos

Legacy systems often trap information in isolated repositories, making it difficult to deliver real-time insights and blocking the adoption of modern analytics and AI and Machine Learning capabilities that depend on integrated data.

Ecosystem integration

Increasingly, organisations need to be able to integrate with third parties via APIs in order to perform their core business functions (e.g., integration with exchanges in financial and energy trading).

Regulatory compliance

Regulations may affect a given sector or every organisation across the economy (e.g., UK GDPR), requiring the modernisation of systems and processes.

Customer and staff experience

Customers now expect seamless mobile and web experiences, including personalisation and real-time services; and modern tools can foster greater staff productivity.

Business agility and innovation

A 20-year-old system that delivers value isn't legacy. A five-year-old system that holds you back is. Legacy systems act as a drag on time-to-market, impede the ability of organisations to scale or adapt to new market conditions, and stand in the way of innovative business models.

Contextual and organisational changes

A system may have been developed to fulfil outdated business needs or to suit a specific context that no longer pertains (e.g., a global pandemic). Features, workflows and whole systems can become legacy due to changes in the organisation or in the wider world.

Security compliance

Older systems lack modern security features, making them much more vulnerable to attacks.

Cost optimisation

Low availability of talent with the required knowledge of old technologies may make a system too costly to maintain. In the same vein, as systems move beyond their expected operational lifespan, replacement hardware can be costly to obtain, and maintenance packages increase in cost. Legacy infrastructure is often less efficient, driving higher energy costs.

Regardless of the driver, effective legacy modernisation depends on having a comprehensive view of your organisation's legacy estate – a map that enables you to navigate the iterative modernisation process with confidence. We look at **Mapping your IT estate next.**

Mapping your IT estate

Mapping your IT estate

This capability sits outside of the iterative modernisation process and underpins it. It provides continuous visibility of your estate, helping you identify modernisation candidates and the dependencies that must be considered when implementing change.

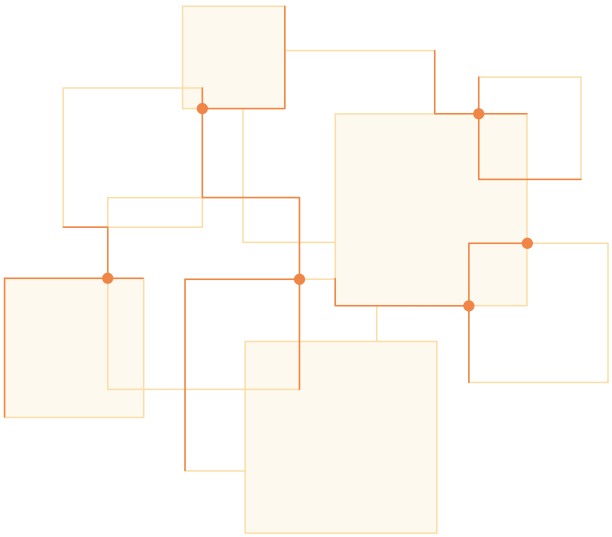
In our experience at Scott Logic, the organisations we work with usually already hold a high level understanding of their technology estate, built up over time through architectural work, documentation, and internal knowledge. This human insight forms one half of the picture and is vitally important, but it is not enough on its own. What it provides is a static snapshot of the estate at a point in time, with an incomplete view of dependencies across the estate.

Modern technologies such as Observability Platforms and Application Lifecycle Management (ALM) tools now provide the other half of the picture. It's thanks to these tools that we can now perceive IT estates as dynamic rather than static, and get to grips with all their moving (and ageing) parts. They reveal discrepancies between the documented view of the estate and its operational reality. For example, tools might surface unexpected connections, dependencies, or components that have become outdated or which internal diagrams have missed.

Through automated monitoring of application lifecycle states, ALM tools provide you with continuous observability. After you define key indicators and thresholds for each application and system in your estate, these tools will trigger alerts, workflow actions or reports when those conditions are met or exceeded.
With all of this real-time data, organisations can now be in a better position than ever to make evidence-based decisions about legacy modernisation. However, as we advise our clients, it's important to remember that ALM tools provide visibility, not judgement. They show the true current state but do not tell you what counts as a problem. Only people who understand the business context can determine that. Insights such as risk, cost impact, or strategic relevance must still come from conversations, experience, and organisational knowledge. Tools help keep the estate map accurate and current, while human insight ensures that map is meaningful, connected to strategy, and used to guide decisions.

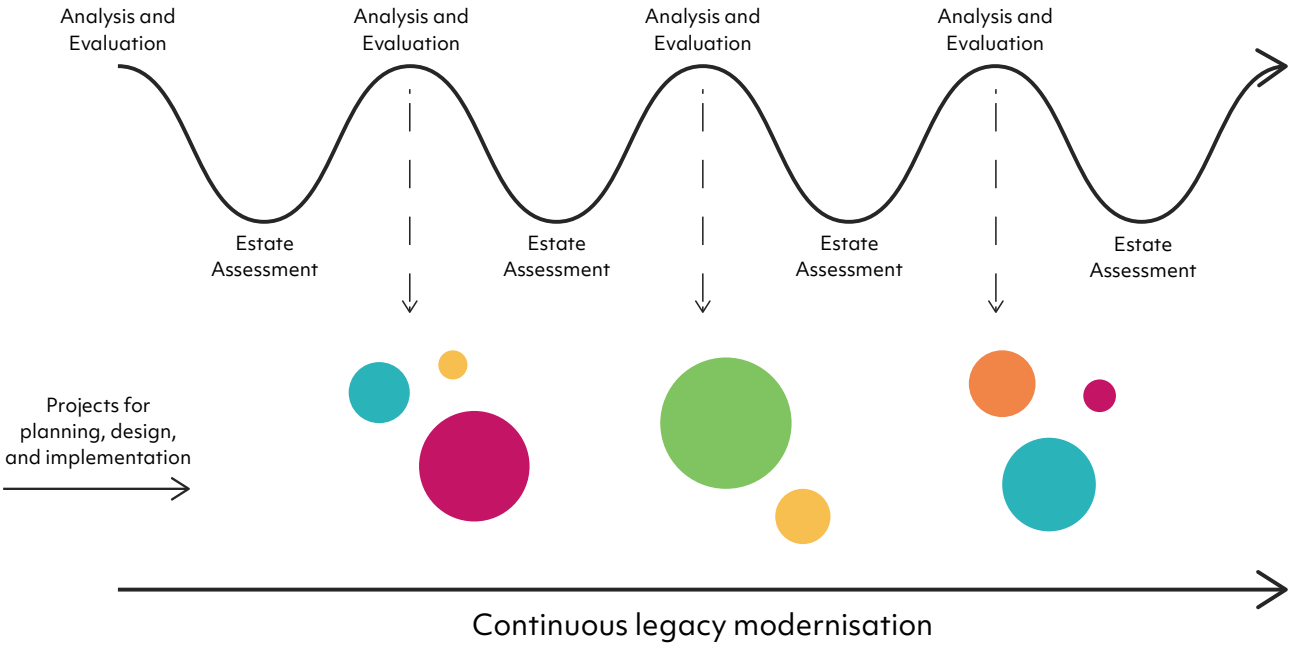
Establish a baseline map...

As a foundation for iterative legacy modernisation, we recommend the creation of a baseline map. This is a one-off task to chart your IT estate comprehensively, including its network of relationships and interdependencies, and its underlying infrastructure. With all the human insight distilled, ALM tools can fill any gaps and then help ensure the map remains accurate and up to date. The map will need iterating over time as new systems are added to the estate, but this will not require a repeat of this mapping exercise; rather, it's a maintenance task.



...then evolve it continuously

As well as enabling day-to-day defect resolution in response to triggers, ALM tools can be used to run estate assessments on a regular cadence – say, every six months. Through these assessments, candidate components can be identified and moved into the iterative modernisation cycle for **analysis and evaluation**. For any given modernisation candidate, the outcome may be a very short project or one lasting many months.



As a result, it is entirely possible – and entirely manageable – to have numerous modernisation projects of varying complexity and duration running in parallel. The projects set in train following an estate assessment may well still be underway when the next one comes around. At Scott Logic, we therefore recommend to our clients that they decouple the cadence of estate assessments from the planning and implementation of modernisation projects.

While this proliferation of projects may sound daunting, they will be of relatively small size compared to the 'Big Bang' projects of old and they will require smaller investments. In-house capacity may become an issue if the estate requires multiple modernisation projects at the same time, or if a given project is of greater complexity or higher time-sensitivity than usual. In instances like this, it's possible to bring in third-party burst capacity for short-term, high-intensity work.

Analysis and Evaluation

Analysis and Evaluation

Once a candidate for modernisation has been identified, the analysis and evaluation phase involves an initial investigation into the problem space, an assessment of the best modernisation option, and the creation of a business case to win stakeholder approval.

Initial triage

For each modernisation candidate, we recommend that you begin the process by bringing together the right stakeholders from across the organisation to carry out initial triage. This allows you to assess the business-criticality of the change and to determine its urgency (e.g., due to an imminent compliance deadline or the termination of a support licence).

It also allows you to ask some fundamental questions about the candidate for modernisation: What business problems is the system or component intended to solve? Are there opportunities to streamline the workflows and processes carried out by this legacy component? As we advise our clients, modernisation isn't just a technical upgrade; it's a chance to align technology with optimised business processes. So, at a high level, you can begin to identify pain points in current processes and flag areas where modernisation could enable automation or new workflows.

It's important at this stage to gain a high-level understanding of the technical landscape, the dependencies, and the key risks. Tools like Application Lifecycle Management platforms will provide much of this information, giving you visibility of the technical architecture, the age and support status, upstream and downstream systems, third-party integrations, and any known security, regulatory or operational risks.

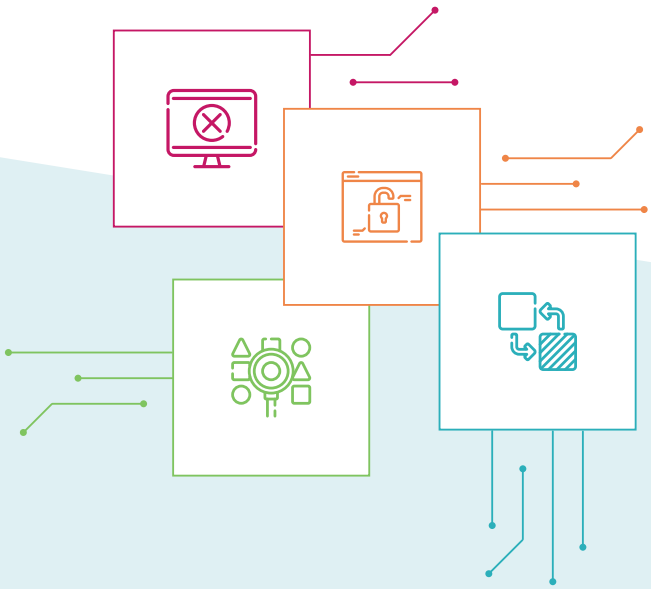
At this early stage, you can also gain a snapshot of the current operating cost of the modernisation candidate along with the maintenance effort required, including whether specialist skills are involved. Opportunity cost can also be considered by taking an initial view on whether the modernisation candidate is blocking innovation or standing in the way of achieving business goals.

The outcome of this initial triage is a set of hypotheses which you can aim to validate or disprove iteratively during the rest of this phase.

Assess options

Based on the initial triage, you should have the information you require to decide on which modernisation strategy is the most appropriate. Within a large IT estate, different systems – and even different components within the same system – will suit different strategies.

Let's look at those strategies next



Scott Logic's 4Rs of Legacy Modernisation

Specific to the context of cloud migration, AWS has its 7Rs which outline the seven most common strategies you can adopt (Retire, Retain, Rehost, Relocate, Repurchase, Replatform, and Refactor)[1]. At Scott Logic, we see this as a useful analogue for the legacy modernisation strategies available to you, but we simplify them down to the following four:

Retire

Remove unused or redundant applications to reduce cost

Retain

Keep application as-is

Repurchase

Replace with an off-the-shelf alternative ("drop and shop")

Rework

A spectrum of choices that are open to you when the other options are not right

The spectrum of Rework

Finding again a useful analogue in the 7Rs, we at Scott Logic describe the spectrum of choices within Rework as:

Rehost

Lift-and-shift the application without code changes

Relocate

Move an entire virtualised environment unchanged

Replatform

Make minor optimisations while migrating (e.g., change the database)

Refactor

Redesign the application in its entirety

You are offered binary choices by Retire, Retain and Repurchase. "Binary" by no means "easy", and all these strategies can require significant planning and careful implementation. However, we find that organisations struggle most with the spectrum of strategies that fall within Rework, and it's here where modernisation projects most often fail to deliver.

Adding to the complexity is the fact that a single strategy might not be enough for a given system.

As part of the iterative and incremental approach to legacy modernisation, you might find yourself Rehosting, Relocating, Replatforming, or Refactoring different components of the same system as it progresses through the legacy modernisation lifecycle.

How best to navigate this spectrum of Rework is the focus of this guide, along with the **technical strategies** and **approaches** that you can adopt.

[1] About the migration strategies, AWS. *Earlier, in 2010, Gartner launched 5Rs:* Migrating Applications to the Cloud: Rehost, Refactor, Revise, Rebuild, or Replace?

Decompose the problem

Before you can make your business case and secure stakeholder buy-in, you need to have a good understanding of the scale and scope of the proposed modernisation work. A more detailed Planning and Design phase comes next; at this stage, you need to have resolved enough uncertainties and gathered sufficient information to support your business case.

At Scott Logic, we always recommend that you bring the right stakeholders "into the room" for this decomposition work, leveraging their deep domain expertise to gain a holistic understanding of the problem space. Depending on the component or system in question, these stakeholders might include software engineers, business analysts, UX designers, operations managers, compliance specialists, and legal experts, among others. As we described in the section on **Drivers of legacy modernisation**, a key input will come from those with responsibility for cross-organisational architecture, who will also bring knowledge of the priorities and principles of the technology strategy. They will act as translators between business, product, design, delivery and engineering so that impacts and trade-offs are understood across disciplines.

It's crucial to avoid focusing solely on the technical aspects. To gain a holistic understanding, we recommend decomposing the problem by analysing it through three lenses; these are set out next, along with some indicative activities.

People

Roles and responsibilities

- Identify which teams own different parts of the legacy system (development, operations, support) and the corresponding business processes.
- Identify any skills gaps in your workforce, both in relation to the legacy system and the targeted technologies.

Stakeholder mapping

- Map out all those people who depend on the system, including internal users, external customers, partners and other third parties.
- Assess the impact of the modernisation on their user experience and operational workflows.

Change readiness

- Gauge the cultural readiness at your organisation for any new ways of working that the modernisation work will introduce (e.g., DevOps, leveraging AI).
- Capture at a high level any training or communication requirements that can be discerned at this stage.

Continued on next page

Processes

Business architecture

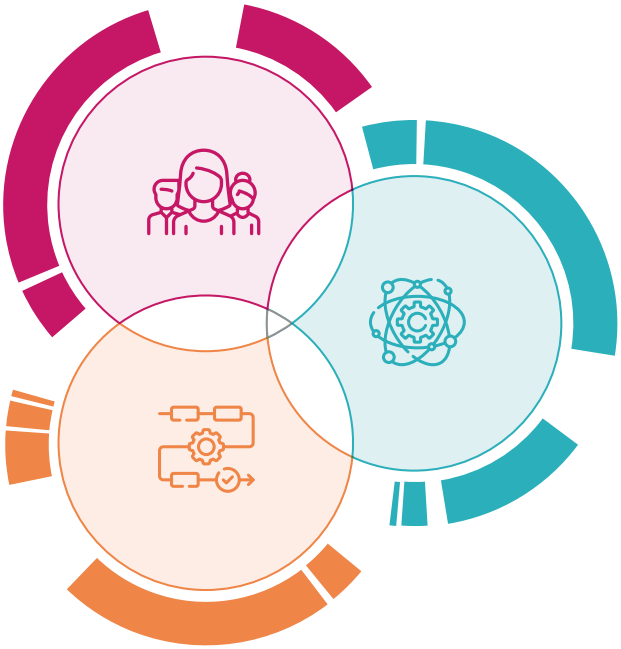
- Map out all of the current business processes that the legacy system supports.
- Identify any inefficiencies or manual steps that the modernisation could eliminate.

Operational workflows

- Document how the system interacts with day-to-day operations (e.g., batch jobs, approvals).
- Look for opportunities for automation or streamlining.

Compliance and governance

- Check that any envisaged process revisions will meet regulatory requirements.
- Plan for auditability in the new environment.



Technology

Infrastructure architecture

- Identify where and how the system runs today, and which characteristics limit scalability, automation, or portability.
- Map how the system connects to other systems and where tight coupling or implicit dependencies will constrain incremental change.
- Assess how resilience, security, and observability are achieved today, and where operational knowledge is embedded in people rather than the platform.

Data architecture

- Map data domains, dependencies, and migration challenges.
- Identify any data quality and portability issues that will need to be resolved.

Application architecture

- If the modernisation candidate has a monolithic structure, break it down into logical modules or services.
- Classify the modules by complexity and business criticality.

Modernisation units

- Look for opportunities to modernise the modules iteratively and incrementally, and group them into units for phased migration.
- Identify the best-suited modernisation strategy for each unit.

The outputs of this decomposition exercise will be inputs that underpin your business case, demonstrating that you have investigated the problem space sufficiently and can describe the impact of the proposed modernisation on people, processes and technology. However, it's important that you articulate the business case overall in business terms, rather than in technical terms.

Secure stakeholder buy-in

It's the job of executive stakeholders with budgetary decision-making powers to prioritise business outcomes over technical considerations. For this reason, we always recommend to our clients that they shape an overarching business case narrative which foregrounds the link between the proposed modernisation and corporate strategic objectives.

With this in mind, you should select value drivers and metrics that the stakeholders will care about, which quantify organisational impacts such as:

- **Cost savings** (e.g., reduced maintenance requirements, lower licence fees)
- **Risk mitigation** (e.g., regulatory or legal compliance, enhanced security, strengthened operational resilience)
- **Enhanced customer satisfaction** (e.g., more frequent product/service releases, improved user journeys)

During implementation, you can then maintain momentum through regular, transparent reporting that makes risk, progress and value visible.

HiPPOs and Leaders

There's always the risk of the HiPPO Effect (the Highest-Paid Person's Opinion) or corporate politics derailing your modernisation proposal. However, the Analysis and Evaluation approach we recommend will help to mitigate this in various ways.

By bringing together stakeholders from across the organisation throughout this phase, you will have demonstrated cross-organisational buy-in for the modernisation. Your data-driven analysis to decompose the problem will have generated facts that will be difficult to argue against based merely on personal preference or pet solutions.

In addition, you can invert the cost of doing something by articulating the cost of doing nothing – the growing cost of maintenance, as well as the reputational costs of downtime and other service disruptions.

Your business case can instead appeal to the bravery of your organisation's leaders. Traditionally, legacy modernisations have been seen as disruptive and unpopular, discouraging leaders from putting themselves forward as champions and sponsors of these initiatives. As we've explained, by reconceiving legacy modernisation as a continual process, it's much easier to frame it in strategic terms that appeal to leadership; proactive modernisation demonstrates foresight and becomes a strategic initiative to future-proof the organisation. Suddenly, a footnote on your CV becomes a proud part of your legacy.

Key business case components

Your business case should present a compelling narrative that resonates with the executives in your organisation, presenting a succinct argument for the modernisation. It should be accompanied by a more detailed supporting document that presents the outputs of your analysis and evaluation, written for finance and technical teams.

A tried-and-tested structure is as follows:

- **Executive Summary** – a clear, concise, business-focused rationale
- **Current State Analysis** – an overview of key risks, costs, and inefficiencies
- **Modernisation strategy** – the recommended modernisation strategy, such as the ones in Scott Logic's spectrum of Rework
- **Cost-Benefit Analysis** – including direct and indirect benefits, and the cost of doing nothing
- **Risk Assessment** – the risks of the proposed modernisation work, including recommended mitigations
- **Implementation roadmap** – a visualisation of the high-level timeline and milestones
- **KPIs and success metrics** – how success will be measured (e.g., cost reduction %, uptime, speed to deploy)

Planning and Design

Planning and Design

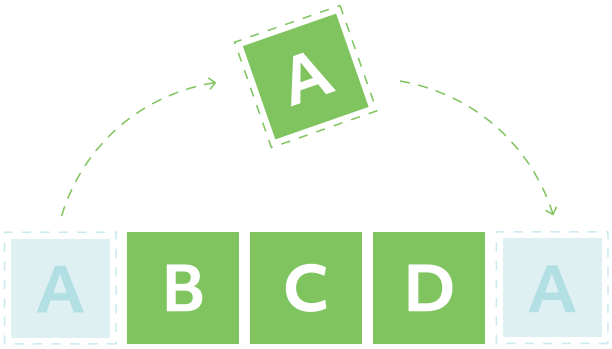
With stakeholder buy-in secured and investment committed, the next step is to plan and design the implementation. The outputs of the Analysis and Evaluation phase will provide many of the inputs you need. With all the focus on people, processes and technology up to now, it's important at this stage not to forget non-functional requirements (NFRs); at bare minimum, the modernisation project should maintain the standards of performance, security, accessibility, and usability (among other NFRs), and should ideally aim to improve them all.

The purpose of this guide is not to set out how to plan and design a software delivery project, so we won't cover that here. However, it is one of the purposes of this guide to assert that planning a legacy modernisation project is critically important.

It's true that most greenfield software projects require less up-front planning and design before implementation gets underway. But how you implement a legacy modernisation project is intrinsically different, and this means that more planning and design are required.

As we covered under **Drivers of legacy modernisation**, the business drivers are different. In greenfield projects, the primary driver is often revenue generation, whereas legacy modernisation projects tend to focus on cost reduction, operational efficiency, and risk mitigation. This difference impacts decision-making and prioritisation, with greenfield product roadmaps usually being influenced by marketing and commercial strategies. In contrast, a key milestone on a legacy modernisation roadmap might be the date when new regulations come into force. (That said, as mentioned before, the business case should nevertheless be articulated in commercial rather than technical terms.)

Self-evidently, the as-is state is more complex on a legacy modernisation project. As we described in the **Decompose the problem** section, a legacy project requires you to analyse the as-is state through the lenses of People, Processes and Technology. This all needs to be considered in charting the route to the to-be state – and this route may often appear circuitous compared to a greenfield project's straight line towards a minimum viable product.



For example, in our experience at Scott Logic of client legacy projects, the sequencing is not always as obvious as it first seems. Let's say you have four interdependent systems in scope for the modernisation project and that, at first, the logical sequence appears to be A, B, C, then D – chiefly, because system A is expensive to maintain and the business wishes to offload it first. However, you may find that offloading system A in the least disruptive manner requires 'enabling changes' first to systems B, C and D. On another project, you may discover that there's unexpected early value to be unlocked by prioritising work on, say, system C ahead of the others.

It's this quality of legacy modernisation projects in Scott Logic's category of Rework which makes it sensible to take an incremental and iterative approach. Planning and implementation within each iteration can be along agile lines, but sufficient up-front planning is required to work out the optimal phasing of those iterations. 'Optimal' being the apt word, here; there's usually so much business architecture at play in legacy modernisation projects that there's often not a 'right' sequence – you have to aim for the sequencing that's as good as it can be in the constraints you're working within.

Create a Test Strategy

Testing is of such critical importance throughout the implementation phase that you should not commence implementation without a test strategy in place. It enables you to shape your approach to risk reduction proactively, rather than reactively after implementation is underway. In addition, knowing what you will test, and how, can have an impact on architecture and design decisions, so it's important to gain early clarity on your approach.

It's not that Scott Logic advocates the waterfall approach of assuming that most risk mitigation can be achieved through up-front design; we know that the test strategy is likely to evolve during implementation, particularly over the course of longer projects. Instead, what we're advocating is that you should establish the core tenets of your testing framework and strategy before implementation begins. This will set you up for success.

Your organisation's test engineers will know what the core elements of the test strategy should be. So that you can have useful discussions with them about it, here are a few key components that we recommend you look out for:

Alignment to business goals

The strategy should show a clear understanding of the business processes supported by the system – the 'happy' path and the 'non-happy' paths. By extension, it should show an understanding of which workflows and processes are a priority or even business-critical, and whether any major changes in input or output are likely to be implemented. It should also advocate prioritising tests by risk, not only by requirements coverage.

Data management

The strategy should demonstrate a clear grasp of how data will be managed during implementation. For example, it should state whether most of the testing will be validated and verified in test environments, or whether some testing will be achievable in the production environment (and if so, what caveats and mitigations need to be in place around data protection). It should indicate 'mocks' will be used (i.e., a test double that replaces a real dependency, simulates its behaviour, and records interactions). In relation to data migration, it should recommend that data is cleaned prior to migration so that it can be mapped efficaciously into the new system or workflow.

Regression testing

The purpose of regression testing is to check that new changes haven't broken anything that used to work and this, self-evidently, is of critical importance on a legacy modernisation project. At minimum, the strategy should advocate regression test coverage of high risk and heavy traffic integrations and user journeys. As part of the regression pack, it should recommend the use of test automation (which speeds up repetitive checks and improves consistency) and contract testing (which checks that two systems are 'talking' to each other in the intended way). The regression suite should be established and optimised before any data migration takes place. Given that legacy modernisation offers the opportunity to streamline processes, it's important to remember that, by design, the systems might not carry out processes in the same way.

Failover and Disaster Recovery

Resilience is as important as functionality when it comes to legacy modernisation. The strategy should set out a systematic approach to failover and disaster recovery (DR) testing. Ideally, this should be done first in test environments to draw out any "gotchas". If it's possible to put the legacy system through a failover and DR test first, the process and outputs from this can be mapped into the equivalent tests in the new system. Both failover and DR testing should happen before declaring the project "done", in order to provide confidence that the new environment can handle unexpected failures.

Non-functional testing

Non-functional testing evaluates the qualities of a system – performance, usability, security, reliability and operability – focusing on how well the system works rather than what it does. Because modernisation often introduces architectural and behavioural change, non-functional considerations become central to the success of any migration. At a minimum, the modernised system should meet the legacy system's existing Non-Functional Requirements (NFRs); wherever possible, it should improve on them.

In order to achieve this, you need a clear understanding of the as-is state across all relevant non-functional domains. This baseline should be informed by real usage data so that NFRs and Service Level Objectives (SLOs) reflect the actual needs and behaviours of users rather than assumptions. The test strategy should set out the tooling, metrics, and methods to be used for non-functional testing, while also defining the scope and limits of each testing level so that responsibilities and expectations are unambiguous.

Observability and operability also need to be designed in from the outset. Retrofitting them later is risky and costly. Modernisation should assume that failures will occur, and the system needs to be capable of detecting, diagnosing and recovering from those failures quickly. By embedding observability early and ensuring that non-functional behaviour is continually verifiable in production-like conditions, organisations can increase resilience, reduce operational overheads, and create systems that are more predictable, supportable, and sustainable over the long term.

Performance

Performance metrics are essential. Modernisation offers a key opportunity to streamline and improve system performance, but "performance" can span many qualities depending on the system's purpose. Before migrating, it's important to understand how the legacy system manages different data types, behaves under peak and sustained loads, and handles expected data volumes, as this helps reveal bottlenecks and opportunities for improvement.

It's also the right point to assess scalability: many modernisation efforts begin because the legacy system can no longer support organisational growth. Cloud migration may increase agility and scalability, but these assumptions still need to be validated through testing.

Accessibility and usability

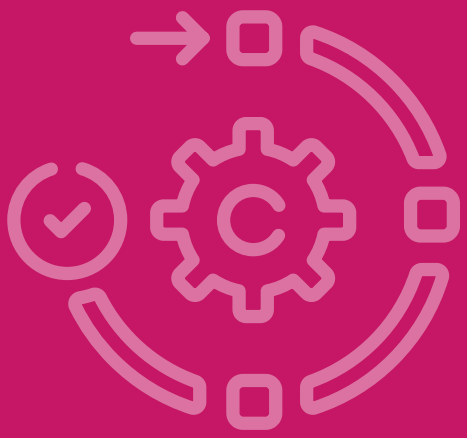
Accessibility and usability are important considerations, even if they may rank slightly below performance and security. The test strategy should advocate validating that real user needs – accessibility, clarity, reliability in the moments that matter – are met or improved.

Security

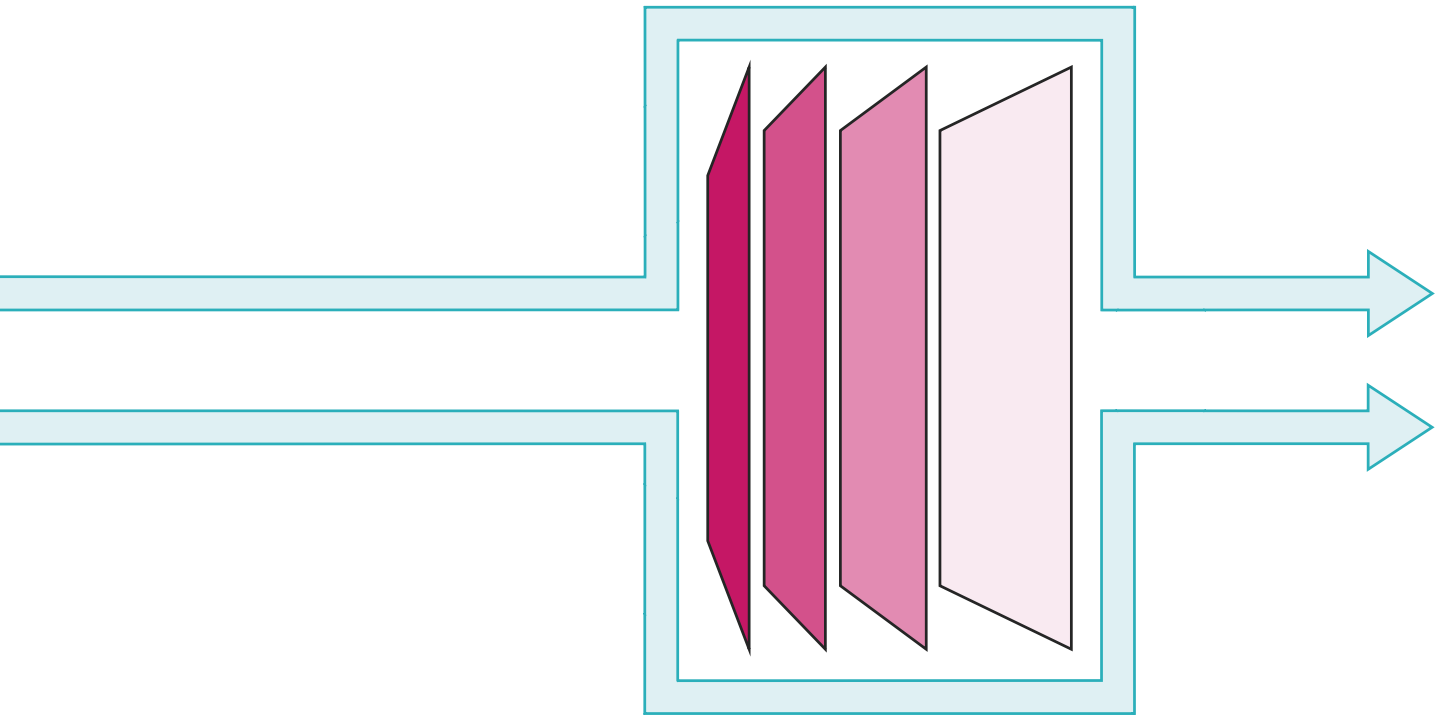
Security testing is another essential element of the test strategy. It involves understanding potential vulnerabilities, how they can be mitigated, and how they should be tested. This requires reviewing the current security processes, defining how they will be validated in future, and considering how the security posture may need to change depending on the chosen modernisation approach.

Compatibility

Another non-functional factor that can be particularly challenging is compatibility. Some legacy systems were designed for, and may still depend on, outdated hardware or software. It is therefore essential to test thoroughly that the modernised system will function correctly on the newer platforms before migrating any production workloads.



Implementation



Implementation

Any modernisation project that falls within the spectrum of Rework should take an iterative and incremental approach. This allows you to reduce risk, unlock early value, and align your technology upgrades with business priorities. But what are the technical strategies that support Rework?

Coexistence strategies

A coexistence strategy is an approach where the legacy system and new components (which often form a hybrid architecture rather than a single replacement system) operate side by side for months or even years. For complex systems, this allows you to migrate and test functionality incrementally while maintaining business continuity.

In this section on Implementation, we will describe some of the best-established coexistence strategies, all of which Scott Logic has used to help modernise clients' legacy systems. We'll make reference in this section to technical approaches that can be used to implement these strategies and then describe them in more detail in the next section.

Continued on next page

Strangler Fig Pattern

What it offers

This strategy allows you to replace large monolithic legacy applications incrementally with smaller, easier-to-manage services and applications. This often involves writing a façade such as an API Wrapper in front of (or over the top of) the legacy system, then redirecting traffic to the new services once they have been tested and signed off. Like its botanical namesake, the Strangler Fig Pattern gradually builds new functionality around the old system, using its structure for support until the legacy code can be safely retired. Many of the approaches outlined below either feed into this pattern or are variations of it.

When to use it

This strategy is best applied when you want to transition a business-critical system from monolithic architecture to microservices or hybrid architecture, or when you are dealing with web-based monolithic applications.

Technical approaches

- API Wrapping
- Event-Driven Architecture
- Microservices
- Data Migration

Risks and challenges

- There's a risk of missing key journeys when you're making changes and introducing new features at the same time as replacing the processes carried out by the legacy system. Additional mapping and business logic may be needed.
- With complex systems, there's a risk of routing traffic to the wrong place – i.e., directing new inputs to legacy databases or pulling old data from legacy databases and presenting it as new.
- Maintenance of two environments can result in divergence and added complexity, as well as added costs. If major business process changes are expected during the project that affect both the legacy system and the new one, they will need careful planning and scheduling.
- The overheads of running and maintaining two systems can outweigh the benefits.
- If there is an urgent need to decommission the legacy system due to a regulatory change, this might not be a suitable strategy. In such instances, we would advise clients to consider a "big bang" replacement or repurchase.

How to do it safely

- The team needs clarity on what is migrating when, with all dependencies mapped and any changes to data structures or quirks in the legacy system or external connections understood.
- Robust testing is required, with regression packs built into deployment pipelines. In 'lift-and-shift' instances, user journeys should be easy to map and test as per the existing system.
- The test strategy should include:
  - Integration and regression testing to ensure that the new system behaves in the expected manner.
  - Failover testing and Disaster Recovery testing on the new services prior to going live.
  - Contract testing of any API integrations between the proxy and the old/new systems to highlight when breaking changes are made.
- Decommissioning must be planned from the outset. Although legacy systems are usually kept running during migration to provide business continuity, there must be clear criteria for when they can be retired and a plan for what happens if momentum or funding stall. Before any decommissioning takes place, it's vital to secure confirmation from the business that the new service fully meets requirements.
- Prior to decommissioning, it is also essential to make sure that all users are using the correct system.

Adapter and Anti-Corruption Layer

What it offers

This strategy allows you to maintain clean boundaries between the legacy system and the new system, preventing corruption of the new system's design. An adapter sits between the old and new systems to translate the legacy models to the new models. As an anti-corruption layer, the adapter prevents changes in one system from affecting the other.

When to use it

This strategy is suited to contexts in which a messy legacy model needs to be kept distinct from the new model to prevent the repetition of old ways (e.g., building 'quirks' into a system just so it can handle a bit of data that was set in a particular way years ago). The strategy is also particularly useful when the volume of change in the legacy system is high.

Technical approaches

- Domain Mapping
- Event-Driven Architecture

Risks and challenges

- This strategy requires additional mapping logic and translation rules, which can become complex very quickly. This can in turn cause a performance overhead, resulting in reduced system responsiveness.
- More maintenance and vigilance are needed to ensure translations are correct.
- There is additional testing complexity due to the need for unit and end-to-end integration tests that attempt to reproduce often difficult legacy behaviour.

How to do it safely

- It's important to keep any adapters and anti-corruption layer code as focused as possible, with clearly documented mapping rules.
- The use of side-by-side testing and 'round-trip fidelity' validates that the key business logic still works by mapping old to new, then new to old, and comparing the results.
- Using a contract testing tool like Pact would be beneficial – defining the contract (i.e., the expected structure and behaviour of requests and responses) for the new system and then testing that it is satisfied when pulling legacy system data.

Layer-on-Top Pattern

What it offers

This strategy allows you to, in effect, 'hide' the legacy system and work on it out of sight at a later point. A new façade or service layer is built on top of the old system. It's a strategy that's essentially a 'flavour' of the Adapter/Anti-corruption Layer strategy described above.

When to use it

This strategy is often used as a first phase before another coexistence strategy is used in the next iteration of a legacy modernisation programme. For example, if the backend of a system will take a long time to modernise, this strategy allows your organisation to present a more modern user interface in the meantime.

Technical approaches

- API Wrapping
- UI Modernisation
- Desktop Interoperability

Risks and challenges

- This strategy potentially involves maintaining two technology stacks with differing needs.
- The new interface will be limited by legacy system constraints (such as performance and latency issues) or require workarounds to achieve all the required goals.
- Depending on the business logic and requirements, this strategy could result in non-functional discrepancies and unmanaged expectations (i.e., users seeing a modern front-end but experiencing the service and performance of an old system).
- There's the risk that the backend is never modernised due to a lack of funding or loss of momentum.

How to do it safely

- The team should ensure that there is a clear plan, including a map of the expected outputs and inputs.
- The layer should be kept as thin as possible, only providing essential functionality without duplicating business logic or adding unnecessary complexity.
- The full user journeys should be tested thoroughly. This may mean initially mocking the responses for the new user interface while it's in development, then testing the integrations to confirm they align with the old journeys.

Vertical Slicing

What it offers

This strategy involves focusing the modernisation on a business capability or a feature. Components related to that capability or feature are modernised together as a slice of functionality, rather than trying to deliver multiple business capabilities at once. This allows value to be realised earlier. Vertical slicing supports the incremental strangling of the legacy system and helps to make data migration more manageable.

When to use it

This strategy is suited to contexts where there are clear feature boundaries or when your organisation wants to become less dependent on shared services.

Technical approaches

- APIs
- Event-Driven Architecture
- Domain-Driven Design
- CI/CD

Risks and challenges

- If the legacy system depends on a lot of shared services, complexity can be increased initially when defining boundaries and understanding when the shared services can be decommissioned.
- When different teams work on separate vertical slices, inconsistencies can creep in. If those slices later need to integrate, aligning them may require extra layers or duplicated logic, leading to unnecessary complexity and over-engineering.

How to do it safely

- It's important to migrate only the data required for the slice.
- This technique should be combined with other strategies such as adapters, anti-corruption layers and strangulation patterns.

Side-by-Side (Parallel Run)

What it offers

By running the legacy and new systems side by side, this strategy allows you to instil user and stakeholder confidence in the modernised system while maintaining business continuity. You can then migrate users and progresses gradually over to the new system.

When to use it

This approach can be particularly useful in highly regulated or compliance-driven environments, where it's essential that the new system works in the expected manner.

Technical approaches

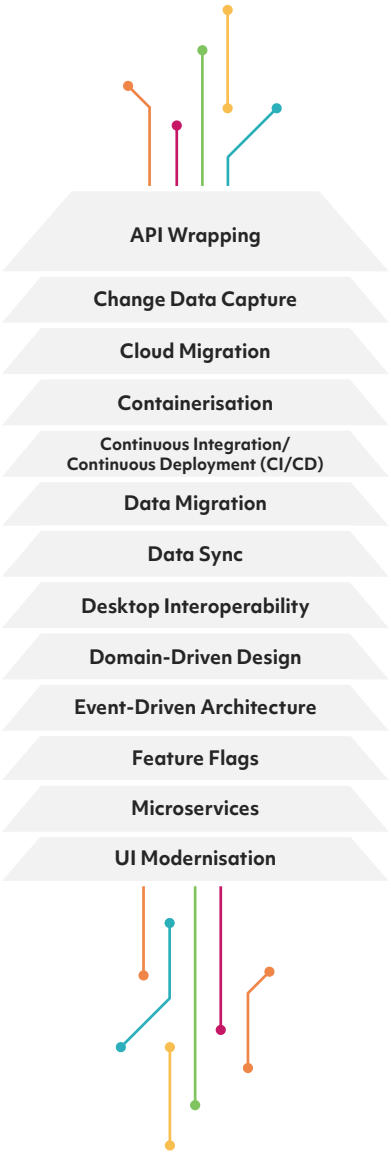
- Data Sync
- API Wrapping
- Feature Flags

Risks and challenges

- It can be costly to run two systems, both in terms of technology and people. This can be manageable in the short term, but you need a clear plan for switching off the old system.
- This approach demands strict control of duplicated inputs, such as bank details or outgoing emails, because entering or triggering them in both systems can lead to inconsistencies and operational risk.
- It can be difficult to identify what is 'noise' and what is relevant for testing; timestamps or different data storage mechanisms can make direct comparisons of results confusing.

How to do it safely

- Parallel running allows real-world (production-like) testing, but you must ensure that any duplicated processes or data during this phase don't end up in the live systems, as this could cause data inconsistencies or corruption.
- You can use asynchronous verification by sending both responses to a queue, where a worker compares them and logs any mismatches. Reporting and metrics should then be updated to ensure these checks are visible and working as expected.



Technical approaches

API Wrapping

API wrapping places a modern API around your legacy system, acting as a translation layer that converts modern requests into a format the legacy system understands, and vice versa. This approach allows you to connect your legacy system to newer platforms and services, supporting automation and unlocking legacy data for use in modern applications – all without needing to rewrite the legacy code. The wrapping can be performed with minimal work using tools such as API Gateways or integration platforms to ensure the wrapping takes minimal effort and is as resilient as possible.

Change Data Capture

Change Data Capture (CDC) allows migrations from legacy systems without significant downtime. In effect, CDC monitors changes in a source database and then delivers them to a target – either in its original form or modified to a new format. Data is often delivered in near real-time. Although this can be complex to achieve, modern database platforms' native transaction logs make this possible with minimal impact to the source. Sometimes, however, it's preferable to stream events instead (see Event-Driven Architecture) as this captures why something changed rather than what changed. Both options make CDC a powerful way to migrate from a legacy system to a new platform, allowing new microservices to consume data from the old system and gradually replace it.

Cloud Migration

This is the process of an organisation moving some or all of its systems over to cloud infrastructure. This can include the migration of databases, files, applications, and containers. Migrating applications requires careful planning to avoid unexpected costs. However, there are numerous important benefits such as greater scalability, cost efficiency, performance, and reliability compared to on-premises deployments. For example, Scott Logic helped a leading investment bank rapidly improve the scalability and resilience of a key trading platform by designing and proving a modern cloud ready architecture. This incremental, low cost migration approach enabled the bank to achieve on demand capacity while maintaining high availability and minimising operational disruption.

Continuous Integration/Continuous Deployment (CI/CD)

Teams and deliverables both benefit from quick feedback loops. Continuous Integration is all about ensuring code is merged early through regular code commits which trigger automated builds and automated testing to provide quick feedback. This highlights merge issues, reveals dependencies, and detects regressions quickly – all with the aim of improving code quality and accelerating delivery. Continuous Deployment extends this notion to push "good" releases into the release cycle, automatically deploying to one or more environments when prescribed checks have been completed.

Data Migration

In many modernisation strategies, data needs to be separated from monolithic sources and moved into domain-specific stores. This enables teams to choose technologies that best fit the shape and purpose of each dataset. However, this shift introduces challenges: data migration becomes complex because structures often change, and large, unwieldy datasets add further difficulty. During this process, it's critical to sanitise data – validating its quality to avoid the 'rubbish in, rubbish out' problem – and to question whether all the data is still needed (the safest migration is one that does not involve any actual migration!). It's important to apply a risk-driven testing approach to data migration – combining reconciliation, representative user journeys, and contract-style checks to ensure that critical workflows continue to function and that data behaves consistently before and after migration.

Using a data extraction and integration strategy allows data from legacy systems to be lifted and exposed before integration into a centralised environment. This creates the foundation for adopting modern data architectures such as data lakehouses. One of Scott Logic's Principal Architects, Matt Richards, expresses the benefits of lakehouses pithily, explaining that they, "resolve the longstanding trade-off between flexibility and reliability by combining the open, low-cost storage of data lakes with the transactional integrity and governance of warehouses." This reduces duplication and supports advanced analytics and AI by making both structured and unstructured data accessible and queryable in real time.

Data Sync

This is the process of ensuring that multiple applications, systems or platforms have consistent data between them. When an application is being modernised, different components often need access to the same data. This data must stay accurate or inconsistent outcomes will result. Synchronisation can take several forms – from real-time methods such as event streaming or triggers, to delayed approaches like overnight batch jobs, or even database replication. In legacy migration, data synchronisation should be used to keep both the legacy system and target system aligned until the new system can fully take over.

Desktop Interoperability

Desktop interoperability platforms connect an ecosystem of web and desktop applications on a user's desktop, allowing them to communicate and work together seamlessly – sharing data, triggering actions, and streamlining workflows – with minimal modification of the underlying systems. They provide a high degree of personalisation, with the potential to boost productivity significantly. They can also unlock additional value by allowing you to share functionality between different areas of your organisation in a way that traditionally would have required a whole other application. As Senior Developer, Duncan Austin, explains, our work with a market leading energy-trading firm demonstrated a key benefit of desktop interoperability, allowing organisations, "to unlock early business value as large-scale legacy modernisation programmes progress in parallel."

Domain-Driven Design

This approach involves designing systems with a focus on the business domain, making sure that the application's structure reflects the real-world organisation. Use of 'ubiquitous' language (i.e., a shared, consistent vocabulary used by both developers and domain experts) helps ensure the domain experts understand exactly what they are getting, and the development teams to understand exactly what is expected. By enforcing clear boundaries and promoting modularity, domain-driven design reduces complexity and helps to prevent monoliths.

Event-Driven Architecture

With Event-Driven Architecture (EDA), modern services listen to 'events' emitted by the legacy system – for example, 'OrderPlaced' – and then execute processes in response to those events. These new services are not directly coupled to the legacy system, which allows them to be iterated without any modification of the legacy code and with no impact on the system's stability. This enables you to introduce scalable, resilient and performant new functionality incrementally, with the strategic aim to retire the legacy system at the appropriate time. As one of Scott Logic's Technical Principals, James Moore, puts it, "event-driven architecture enables a low-risk, incremental path to modernisation that enhances agility and scalability, without disrupting business-critical systems."

Feature Flags

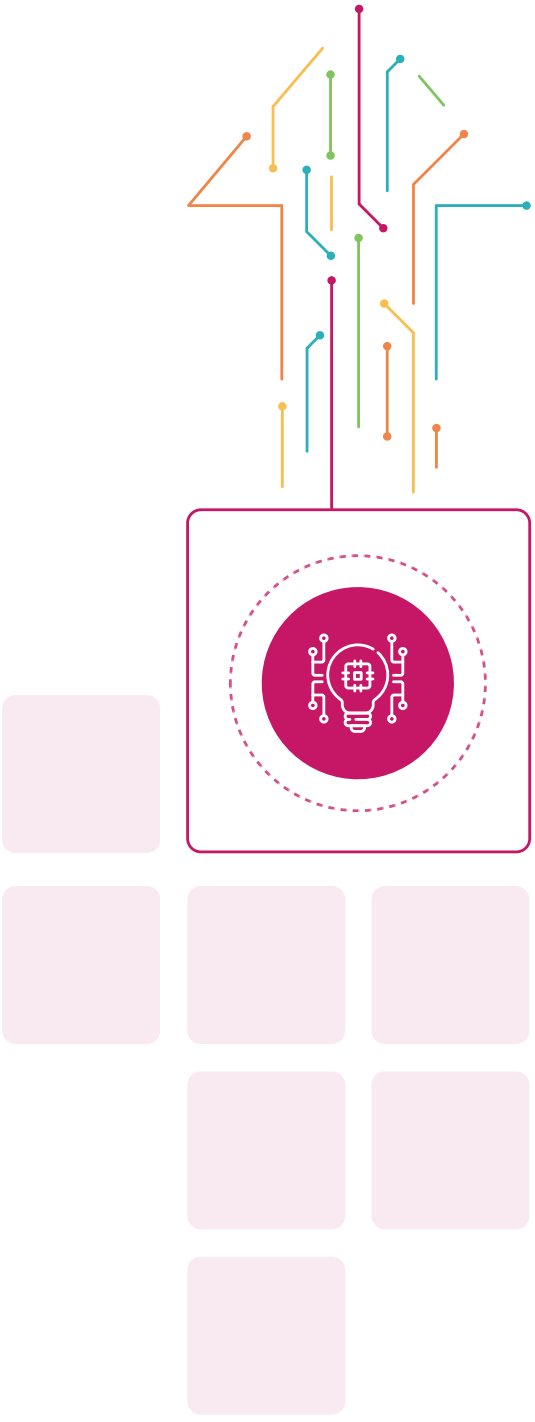
A feature flag (or feature toggle) is a conditional check in the code that controls whether a specific piece of functionality runs. They are commonly used to make refactoring and feature rollouts safer by allowing teams to enable or disable code through configuration. This configuration can be based on permissions, settings stored in a database, or routing rules. It's important to manage the lifespan of feature flags carefully, as leaving them in place too long can introduce unnecessary complexity.

Microservices

A microservice is a small, independent service, focused on a specific business function. For legacy modernisation, microservices make it possible to carve out and modernise individual capabilities safely and iteratively, without disrupting the rest of the system. Microservices communicate with each other, often using lightweight APIs or an Event-Driven Architecture, and can be developed, deployed, and scaled independently. This makes complex applications more manageable, scalable, and adaptable. The size and self-contained natured of a microservice makes it easier to create and maintain. Each microservice is treated as a black box: as long as its external contract (such as its API) remains unchanged, developers are free to modify its internal implementation without affecting other systems that depend on it. With a robust testing approach, and a thought-through release process, microservices can be updated with speed and confidence.

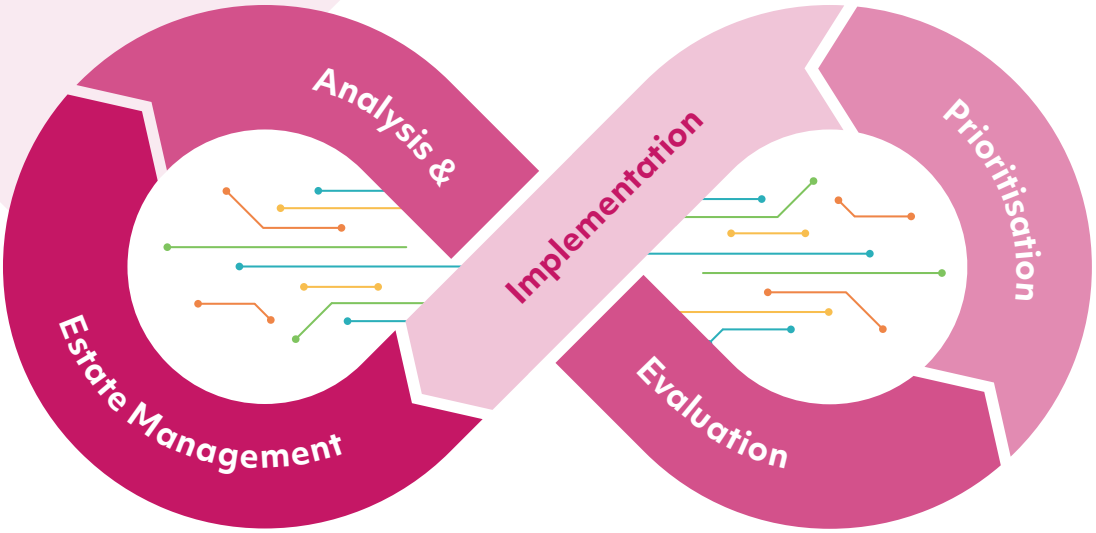
UI Modernisation

UI modernisation involves updating an interface to modern standards. This can include improvements to usability and accessibility, making the interface more responsive, or redesigning it for mobile use. In some cases, modernisation may require completely rethinking how users interact with the application. The desired outcomes are to enhance user satisfaction and productivity while improving maintainability. UI modernisation is often a stepping stone towards modernising a wider system and may act as a precursor to, or follow, the modernisation of the backend components that sit behind it.



Go back to the start

In the new world of iterative and incremental legacy modernisation, your regular estate assessments will be creating a backlog of modernisation candidates prioritised by business needs. So, once you've finished the implementation phase of your current project, you can get underway with the Analysis and Evaluation phase of the next top-priority modernisation candidate – whether it's focused on another component of the system you've just been working on, or a different system altogether.



Know when you're done

Having iterated through all the incremental phases of a complex legacy modernisation programme, there comes the point when a system is ready to switch off. What happens then?

With incremental modernisation, the final 20% of the system often lingers. When you're planning a phased modernisation programme, it's therefore important to define clear goals at the start and establish metrics that will show when the system is ready for retirement.

When you reach that point, the retirement process is fairly simple, but it's important to get it right. You need to check that all the data is where it should be – whether that means archived, deleted, or safely stored in the new system. You need all the stakeholders involved to sign off and confirm everything is ready. Double-check that users, processes, and data have moved over successfully, and that the core business functions work as expected. Finally, review any dependencies you mapped earlier to ensure nothing's been missed, and retire the old system in the agreed sequence.

Contributors to The Legacy Modernisation Guide

Sam Perridge, Energy, Commodities & Utilities Director

Tim Addison, Technical Principal

Mari McMahon, Lead Test Engineer

Colin Eberhardt, CTO

Oliver Cronk, Technology Director

Paul Dykes, Communications Lead

James Heward, Principal Architect

Rich Rogers, Test Principal

Josie Walledge, Delivery Principal

Rob Newsome, Principal Consultant

Duncan Austin, Senior Developer

Greg Lewis, Developer

Nick Hume, Delivery Principal

James Moore, Technical Principal

Catherine Pratt, Delivery Principal

Ready to unlock agility and future-proof your organisation?

For years, we've helped organisations take a pragmatic, iterative approach to modernisation, reducing risk while delivering early value. With our support, you can strengthen resilience, cut costs, and create the flexibility you need to stay ahead of change.

If you'd like to explore how an incremental modernisation strategy could work for your organisation, we're always happy to talk.

Contact our Sales team:

+44 333 101 0020

sales@scottlogic.com

Scott Logic

6th Floor, The Lumen
St James Boulevard
Newcastle Helix
Newcastle upon Tyne
NE4 5BZ

+44 333 101 0020

scottlogic.com