A KEVIN DUNCAN PLAYBOOK

The Strangler Method

Why the safest way to modernise a legacy system is never to replace it all at once, and what one bank's £330m mistake proves about the alternative.

THE CORE PROBLEM

The big-bang rewrite is the most expensive way to fail

Every legacy system eventually tempts someone into the same plan: switch it all off on a Friday night, switch the new one on over the weekend, and hope.

In April 2018, TSB Bank attempted exactly that: a single-event migration of 5.2 million customer accounts onto a new core banking platform. The data migrated successfully. The platform did not. Customers were locked out of banking services for weeks. The Financial Conduct Authority and Prudential Regulation Authority later fined TSB £48.65m for failing to organise and control the migration adequately, on top of over £300m in total costs the bank itself later disclosed, and the departure of senior leadership.[1]

None of that happened because modernising the platform was the wrong call. TSB's systems genuinely needed replacing. It happened because the plan had no reversible step in it. One cutover, no rollback, five million customers on the wrong side of it if anything went wrong. Something did.

> **There is an alternative, and it isn't "move slower."** It's a pattern for replacing a legacy system component by component, with the old and new running side by side throughout, so that every single step is small enough to test, and reversible enough to survive being wrong.

WHERE THE NAME COMES FROM

A vine, not a wrecking ball

The pattern takes its name from an observation by software consultant Martin Fowler in 2004, after watching strangler fig vines in the rainforests of Queensland. A strangler fig germinates high in the branches of a host tree, sends roots down to the ground, and slowly grows around the host over years. Eventually the host dies and rots away, leaving the fig standing in roughly the shape of the tree it replaced.[2]

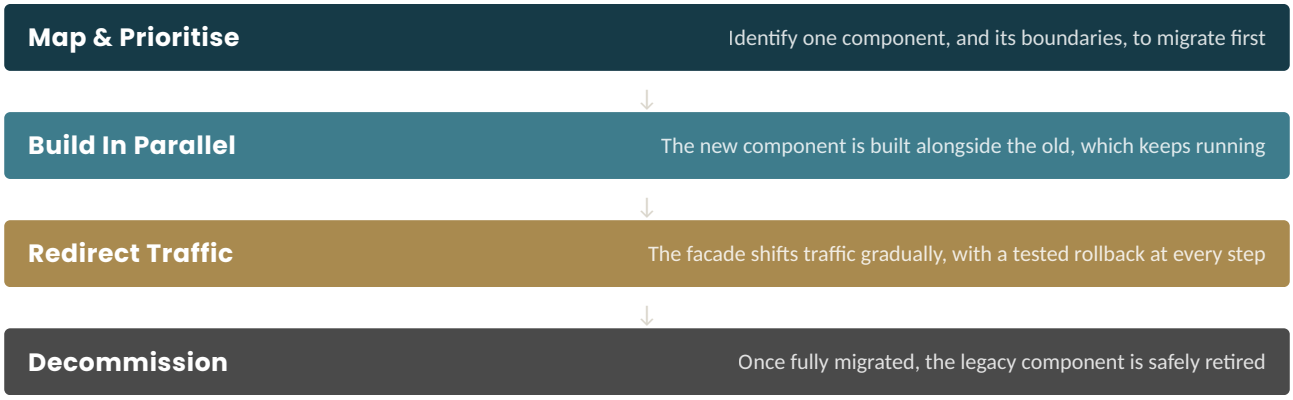
Fowler saw the same shape in good legacy migrations: build the new system around the edges of the old one, route a little more traffic to it as each piece proves itself, and let the legacy system shrink until there's nothing left to decommission. The old system keeps running, and keeps earning its living, for the entire migration. Nobody is ever betting the whole business on one weekend.

> **The mechanism is a facade.** A routing layer, an API gateway or a reverse proxy, sits between users and the underlying systems. Early on, it sends almost everything to the legacy system. As each new component is built and proven, the facade quietly redirects more traffic to it, until the legacy system has nothing left to do.[2]

THE CHAIN

Four phases, repeated per component

The pattern is run once per component of the legacy system, not once for the whole thing. A large system might go through this cycle dozens of times before it's fully replaced.



TWO WORKING PRINCIPLES

Why it works

01 Every step is reversible

The facade can always redirect traffic back to the legacy system if a newly migrated component misbehaves. That single property, reversibility, is what a big-bang cutover cannot offer at any price. TSB's migration had no equivalent fallback once the switch was thrown.[1] A strangler migration can fail a hundred small times without a single customer noticing.

02 Value arrives before the project finishes

A ten-year big-bang programme delivers zero value for nine years and eleven months, then everything at once, on the single riskiest day of the entire effort. A strangler migration delivers a working, modernised piece every few weeks, each one immediately useful, each one de-risking the pieces still to come.

PUTTING IT TOGETHER

Two migration plans, compared

The same legacy core banking platform, migrated two different ways.

BIG-BANG APPROACH

- One cutover event, one weekend
- All 5.2 million accounts move at once
- No meaningful rollback once started
- Zero customer-facing value until launch day
- A single failure affects every customer at once

STRANGLER APPROACH

- Dozens of small migrations, over months
- Accounts move in cohorts, or by product line
- Facade reroutes back to legacy on any issue
- Each migrated component adds value immediately
- A single failure affects one component, briefly

The end state is identical: a fully modernised platform. The difference is entirely in how much risk was on the table on any single day of getting there.

WHERE THIS BREAKS DOWN

Common pitfalls

- **Underestimating the data.** Routing requests is the easy part. Keeping data consistent between an old and new system running in parallel is consistently the hardest part of every phase, and the part most plans underestimate.
- **Choosing the wrong first component.** Migrate something trivial first and you learn little. Migrate the most tangled, business-critical component first and you've recreated the big-bang risk inside a supposedly incremental plan.
- **Never removing the facade's training wheels.** A "temporary" routing layer left in place indefinitely becomes a second legacy system to maintain, on top of the one you were trying to remove.
- **Treating an ambitious timetable as fixed.** The FCA's findings on TSB specifically cited a launch timetable that stayed fixed despite known delays earlier in the programme.[1] A deadline that can't flex is a big-bang risk wearing an agile label.
- **Skipping the rollback test.** A rollback path that has never actually been tested is a rollback path that will fail exactly when you need it, under exactly the pressure that makes failure most expensive.
- **Applying it where it doesn't fit.** Systems that are small, simple, or too tightly coupled to cleanly separate may genuinely need a different approach. The pattern is powerful, not universal.

THE CANVAS

Map your own migration

List the components of your legacy system, and be honest about which is genuinely tangled versus which just feels that way because nobody has looked closely.

COMPONENT	COMPLEXITY	MIGRATE ORDER	ROLLBACK PLAN

Start with the component that teaches you the most for the least risk, not the one that looks most impressive to finish first.

THE AUTHOR

About Kevin Duncan

Kevin Duncan is a product and digital transformation leader with over a decade of experience delivering product strategy and large-scale change across financial services, government and consumer sectors in the UK.

He currently leads product strategy and transformation for the UK mortgage and property ecosystem as Product Director at Novus Strategy & Consulting, work that regularly involves modernising legacy infrastructure inside regulated organisations.

This playbook is drawn from workshops Kevin has run with technology and product teams on de-risking legacy modernisation programmes.

Planning a legacy migration and want a second opinion on the risk?

kevinduncan.co.uk · email@kevinduncan.co.uk · 07891 163404

References

1. Financial Conduct Authority and Bank of England (2022) "TSB fined £48.65m for operational resilience failings," 20 December 2022; TSB's own disclosed costs of over £300m and 80,000 lost customers were reported separately by the bank.
2. Fowler, M. (2004) "StranglerFigApplication," martinfowler.com. Also documented in the Microsoft Azure Architecture Center's "Strangler Fig pattern."