

Prove parity before you rewrite.

Every legacy system carries rules nobody wrote down. Before changing a line, capture what the system actually does, then hold the new version to it.

Toltec engineering

4 October 2026 2 minute read

The short version

- ◇ The running system is the specification. Its documentation is not.
- ◇ Capture behaviour as automated tests before changing anything.
- ◇ Replace the system in slices behind a facade, never in one cut-over.
- ◇ Let AI read and draft. Let the tests decide.
- ◇ Reconcile the data, and keep a way back until sign-off.

The running system is the specification

A system that has run a business for fifteen years has absorbed hundreds of decisions: how VAT rounds, which week a month-end falls in, what happens when a customer code is reused. Most were never documented, and some were never intended. Users rely on all of them.

That is why rewrites from a requirements document go wrong. The document describes what someone thought the system did. The system describes what it actually does.

Capture behaviour first

Michael Feathers called them characterisation tests: tests that record what code does today, rather than what it should do.¹ For a legacy application we build them from real activity, with anonymised orders, invoices and schedules replayed through the old system and every output recorded.

The edge cases matter most: month-end, leap years, British Summer Time, credit notes, zero-value lines and the customer with three accounts. When the old system does something odd, the test records the oddity, and the business decides whether to keep it.

Replace it in slices

Martin Fowler named the strangler fig approach after the vine that grows around a tree until the tree is no longer needed.² A facade sits in front of the old system and routes each request to the old code or the new, so functionality moves across one slice at a time.³

AWS describes the same pattern in three stages: transform a slice, let old and new coexist, then eliminate the old path.⁴ Each slice goes live behind the facade only when its parity tests pass.

Where AI helps, and where it does not

AI is very good at the slow parts: reading unfamiliar code, explaining what a procedure does, mapping an old schema to a new one, and drafting tests from recorded behaviour. It turns weeks of archaeology into days.

It is not a source of truth. Generated code can be fluent and wrong, and it will happily tidy away an oddity the business depends on. So every line is reviewed by an engineer, and every slice is held to the same parity tests as hand-written code.

Reconcile the data

Data moves last and is checked hardest. Row counts, totals per period and checksums are compared between old and new, and the migration is rehearsed until it is boring. The old system stays available, read-only, until the business signs off.

When a slice is done

- ◇ Its parity tests pass against production-like data.
- ◇ The facade routes real traffic to it.
- ◇ Its data reconciles to the penny.
- ◇ The old path stays available until sign-off.

References

-
- 1 Feathers, M. (2004)



*Working No
Effectively
with
Legacy
Code*

-
- 2 Fowler, M.



Strangler martinowler.com, revised 22 August 2024.
Fig

3 Microsoft.



StrangAz Azure Architecture Center.
Fig
pattern

4 Amazon Web Services.



StrangAW AWS Prescriptive Guidance.
fig
pattern