Software

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.
- Prefer slices behind a facade where the system permits it; agree and rehearse the cut-over plan.
- 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 representative activity, with anonymised orders, invoices and schedules replayed through the old system and the relevant outputs 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 when its agreed parity tests and release checks pass. Where a facade is impractical, the migration plan needs a different, explicitly tested route.

Where AI helps, and where it does not

AI can help with the slow parts: reading unfamiliar code, suggesting what a procedure does, mapping an old schema to a new one, and drafting tests from recorded behaviour. The benefit depends on the code, the available evidence and the review it needs.

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. Engineers review the proposed changes, and generated code is held to the same agreed tests and release checks as hand-written code.

Reconcile the data

Data migration follows the agreed cut-over plan. Row counts, totals per period and suitable checksums are compared between old and new, and discrepancies are investigated. The migration is rehearsed, with the retention of the old system and the recovery route agreed before sign-off.

When a slice is done

- Its parity tests pass against production-like data.
- The facade routes real traffic to it.
- Its data passes the agreed reconciliation checks, including financial totals where relevant.
- The agreed rollback or recovery route remains available until sign-off.

References

1. Feathers, M. (2004) *Working Effectively with Legacy Code*. Prentice Hall.

2. Fowler, M. Strangler Fig. martinfowler.com, revised 22 August 2024.

3. Microsoft. Strangler Fig pattern. Azure Architecture Center.

4. Amazon Web Services. Strangler fig pattern. AWS Prescriptive Guidance.

Read the current edition: toltec.io/note-prove-parity.html