PLAYBOOK 5 August 2026 7 min read

How to sequence a legacy system migration

The order of operations that decides whether a migration is uneventful or memorable.

Amit Kumar
Founder, Udyat Technologies

THE SHORT VERSION

Move the least risky thing first to prove the path, cut data over on an agreed date rather than perfectly, and never remove the way back until a full cycle has reconciled.

- Prove the migration path on something low-risk before touching anything critical.
- Data is the schedule. Everything else is more predictable than it is.
- Agree a historical cut-off instead of migrating everything ever recorded.
- Keep a working rollback until one complete business cycle has reconciled.

A legacy migration is not really a technical exercise. Copying data and standing up a new system are the well-understood parts. What determines the outcome is the order of operations, and the order is a business decision more than an engineering one.

The pattern in migrations that go badly is consistent: the most critical system moved first, because it was the one causing the most pain; the data was migrated in full, because nobody wanted to make a decision about what to drop; and the old system was switched off at cutover, because keeping it running cost money.

Each of those is defensible on its own. Together they remove every opportunity to learn cheaply and every route back.

Before anything moves

Unglamorous, and the part most often skipped under schedule pressure.

- ✓ Write down what the system does that nobody has documented — usually held by two or three people.
- ✓ Establish who actually depends on each report, rather than who is on the distribution list.
- ✓ Confirm you can restore the legacy system from backup, by actually doing it once.
- ✓ Find every integration, including the overnight file drop somebody set up in 2016.
- ✓ Agree what 'reconciled' means numerically, before there is any pressure to declare success.
- ✓ Identify the business cycle — weekly, monthly, quarterly — that the migration must survive intact.

The order of operations

Each step exists to make the next one less risky. Skipping one does not save time; it moves the cost later.

01 — Move something that does not matter

A reporting extract, an internal tool, an archive. The point is not the value of the thing moved. It is to exercise the whole path — access, network, data, deployment, monitoring, rollback — while the consequences of getting it wrong are nil.

02 — Move something read-only

A system that consumes data but does not originate it. If it goes wrong, nothing is corrupted upstream, and you learn how the new environment behaves under real load and real users.

03 — Move a self-contained write path

The first genuinely important move: a process that creates data but has few dependencies. Now you are testing reconciliation properly for the first time, on something you can still isolate.

04 — Move the core, in parallel

The system everything depends on, run alongside the old one rather than instead of it. Both take real transactions; the numbers are compared every day until they agree for a full cycle.

05 — Retire, deliberately

The old system is made read-only first, not switched off. It stays reachable for a defined period, then is archived with a documented way to retrieve anything from it.

What to move early, and what to leave until last

Move early

- Reporting and analytics extracts
- Internal tools with a small, tolerant audience
- Systems with few or no upstream dependencies
- Anything already scheduled for replacement
- Archives and historical records

Move last

- Anything finance closes the books with
- Systems a customer touches directly
- The single system every other one reads from
- Anything with a regulatory reporting deadline
- Whatever depends on the one integration nobody understands

Data is the schedule

In almost every migration we have run, the data is what determines the timeline. Not the application, not the infrastructure — the fifteen years of records with inconsistent formats, duplicate customers, and fields that were repurposed at some point for a different meaning.

The decision that saves the most time is the historical cut-off. Migrating everything ever recorded is rarely necessary and always expensive. Active and recent records — typically the last two or three years — are migrated fully, so the new system is genuinely useful on day one. Older records are indexed and made searchable in place rather than re-keyed into the new structure.

Cleaning is a separate decision from moving, and conflating them is a common way to lose a quarter. Move the data as it is, with its problems mapped and visible, then clean it in the new system where you have better tooling. Trying to arrive perfectly clean means the migration waits on a data quality project that has no natural end.

Cutover rules that have earned their place

- Cut over at the start of a cycle, never at month-end or during a peak.
- Have the people who know the business in the room, not only the people who know the system.
- Reconcile before anyone announces success — a matching record count is not a reconciliation.
- Freeze changes to the legacy system for the cutover window, in writing.
- Define in advance what would make you roll back, and who is allowed to make that call.
- Keep the rollback available until one full business cycle has closed cleanly.

On running both systems at once

Parallel running is unpopular because it is genuinely a burden: for a period, some work happens twice, and somebody has to compare the results. It is also the single most reliable way to find the discrepancies that only appear under real conditions.

It does not have to be everything. Running the highest-value or most error-prone transactions through both systems is usually enough to expose the differences that matter, and it keeps the additional load on the team bounded and time-limited.

The period should be defined by a business cycle rather than a number of weeks. If finance closes monthly, parallel running ends when one month has closed on both systems and the figures agree. Ending it earlier because the calendar says so is how discrepancies get discovered by an auditor instead.

A migration nobody outside the project team noticed is not an unambitious migration. It is the whole objective.