Most CRM migrations fail in the places nobody screenshots.

It is rarely the export button, the connector, or even the field mapping spreadsheet. The real trouble starts after go-live, when sales leadership asks where the activity timeline went, support complains that ticket history looks scrambled, and marketing notices that some leads have doubled while others vanished into the void.

A successful CRM data migration is part engineering, part detective work, and part judgment. You are not just moving records. You are preserving meaning, especially meaning that lived in old systems for years. That means you have to handle history correctly, control duplicates without breaking attribution, and map data so the new CRM behaves like the old one, only cleaner.

Below is how I approach the work when the dataset is messy, the schema has evolved, and "just map fields" is not a plan.

Start with intent, not the tool

Before touching a migration tool or writing a mapping document, I lock the intent into plain language. Who needs what, and why?

In practice, "intent" usually breaks into three buckets:

First, operational continuity. Users need to keep working immediately after the migration. They need accounts, contacts, opportunities, cases, and activities to look consistent and behave the same way.

Second, reporting continuity. Leadership expects dashboards to remain trendable. If the last twelve months are broken, adoption suffers fast, even if the CRM is technically "better."

Third, customer history continuity. This is where "history" becomes more than an activity log. It includes notes, emails, calls, assignment changes, status changes, and sometimes embedded document history.

When teams skip this step, they often optimize for the easiest records to move and ignore the records that carry customer memory.

The history problem: preserving timelines without importing noise

History is tricky because CRMs store time in multiple places. There is often a mix of:

- "Current state" fields on objects (like stage, owner, priority)
- Append-only events (like activities, status changes, field change tracking)
- User-generated narrative (notes, call summaries)
- System-generated references (audit trails, workflow logs)

When you migrate, you need to decide what "history" means for each object type.

Activities versus field change history

A common pattern in legacy systems is that sales reps rely on activities as proof of work. If you migrate those activities as-is into the new CRM, the timeline usually holds up. But if the old system tracked additional field changes in a separate audit table, you might be tempted to ignore it because it is "just metadata."

I have learned not to treat that metadata as optional.

Even when leadership does not ask for audit details, reps often ask questions like “When did this opportunity become ours?” or “Why did it switch owners last quarter?” If the timeline is missing the why, the team loses trust in the data.

That said, you also cannot blindly import every audit entry. Audit trails can generate thousands of events per record, and some of them are redundant. Importing them all can turn a timeline into a firehose that nobody reads, and it can slow down page loads and reporting.

The right approach is selective preservation.

For example, if the legacy system logged every time a user opened a record, you probably do not need those events. If it logged stage transitions, owner changes, and status reasons, those are usually worth preserving.

Deduplication can break history if you merge the wrong way

History preservation and duplicate handling are tightly coupled.

When two duplicates represent the same real-world person or account, you usually want to merge them. But how you merge determines whether the history stays intact.

If you merge incorrectly, you might end up with:

- Activities attached to the “loser” record that no longer exists
- Notes that remain orphaned or not visible to users
- Date ordering that becomes confusing because two timelines are interleaved without clear provenance

The safer pattern is to define a merge strategy that keeps the timeline readable. That often means consolidating activities and notes into a single canonical record while storing a reference to where each item came from, at least internally for validation. If your new CRM supports “source record” fields or migration metadata, use them.

If it does not, you can still preserve provenance indirectly, for example by keeping a migration tag in notes or by importing an internal field like “original contact ID” so support teams can trace back quickly.

Time zones and date precision

History issues often show up as subtle timestamp drift.

Legacy systems may store timestamps in local time, or they may store UTC but present in local time, or the connector may interpret strings differently. If you migrate with the wrong interpretation, a call at 3:15 PM can appear as 11:15 PM in the new CRM, and suddenly your timeline is “out of order.”

I usually handle this by doing small-scale pilot migrations on records that have known events. Pick a handful of opportunities and contacts with multiple activities across different days and owners. Compare the ordering and the displayed times after migration.

If you cannot get the time zone behavior aligned in a test, you should not assume it will be fine at scale.

Document history, attachments, and the “where is the file?” moment

Attachments and documents are where history becomes operational.

If the old CRM allowed reps to attach files to notes, tickets, or opportunities, the new CRM will expect similar relationships. If you only migrate metadata or only migrate file names, users will notice quickly.

In one migration I supported, everything looked correct except attachments. The connector imported file rows, but the binary files were missing due to an access token scope mismatch. The team did not catch it until support tried to open a file during a customer escalation. You do not want your go-live “file check” to happen under pressure.

Even when file migration works, there is still the deduplication angle. If you merge two cases that each contain attachments, you need a plan for how the attachments move. Sometimes you merge the cases first, then reattach files to the surviving parent record. Sometimes you migrate attachments with a parent reference that is stable. Either way, decide before you start.

Duplicates: deciding what “same” means and enforcing it consistently

Duplicate handling is usually the part nobody wants to design because it forces you to say what “same person” means when reality is messy.

A CRM can accumulate duplicates for many reasons: imports, manual creation, regional lead lists, different email formats, name changes after marriage, and simply the time gap between data entry and deduplication.

The first step is to classify duplicates by type. Not in a list, but in your mental model:

- Exact duplicates: same email, same phone, same name spelling, same external ID
- Near duplicates: same email but different name formatting, phone variants, swapped first and last names
- Conflicting duplicates: multiple emails, multiple phones, and no clear linkage
- Historical duplicates: duplicates created over time because a person updated an attribute but not consistently

The right merge strategy depends on which bucket you are in.

Use a “canonical key” approach, not just fuzzy matching

Teams often start with fuzzy matching. That can work, but only if you pair it with an authoritative key.

If your legacy system has stable external IDs, use them. If not, email might become the key, but only if you trust email hygiene. If you do not trust it, phone can help, but phone formats vary. Sometimes the best key is a combination, like email plus country, or company domain plus last name.

Where it gets delicate is when you have a real-world scenario like this:

- A contact has two emails because they changed jobs
- The old email still appears in legacy activities
- Marketing uses one email for nurture and sales uses the other for outreach

If your deduplication rule merges based only on email, you might split history into two records even though they represent one person. If you merge based on fuzzy name and company, you might accidentally combine two different people who share the same name at the same account.

The safe approach is to define your canonical key logic with rules that include confidence tiers. High confidence duplicates merge automatically. Medium confidence candidates require review or a “quarantine” step. Low confidence candidates stay separate but get flags for follow-up.

What about accounts that should merge but should not?

Duplicate accounts are a special kind of headache.

An account merge impacts reporting, opportunity rollups, and sometimes the assignment of contacts. If you merge accounts too aggressively, you can erase the boundaries that your team uses for segmentation, like region or subsidiary.

I have seen migrations where two accounts were merged because the company name matched closely, but they represented different legal entities. Opportunities landed in the wrong place after merge, and the pipeline reports were no longer comparable to what leadership expected.

The fix was not just re-running the migration. It was redesigning the account merge rules to prioritize stable identifiers, like tax IDs or external CRM IDs, when available, and treating name-only matching as low confidence.

If your legacy account data includes a stable identifier, even if it is not perfect, it is usually worth investing time to map and validate it.

Practical validation for duplicates

You need a measurable way to prove your duplicate handling is working.

In pilot runs, I look for four patterns:

1. How many merges happened overall, and does that number align with what the business expects?
2. For merged records, do activities and notes appear on the canonical record without obvious gaps?
3. Are owner and status transitions still chronologically plausible?
4. Do reports at the top of the funnel still reconcile within an acceptable range?

Acceptable range depends on your reporting requirements. Sometimes the goal is strict reconciliation for the last quarter. Other times you can tolerate small differences because only current data matters operationally.

What matters is that you decide the tolerance early, because the migration team and the business teams will interpret “close enough” very differently.

Mapping: field-by-field is necessary, but behavior-by-behavior is what makes it correct

Field mapping sounds straightforward until you hit the difference between “value moved” and “behavior preserved.”

Many CRMs have calculated fields, validation rules, required fields, picklist constraints, workflow triggers, and integrations. If you map values into the new CRM but fail to account for validation or transformation logic, the migration can silently skip data or coerce values.

Normalize first, map second

Most messy migrations benefit from a normalization phase.

In this phase, you clean and standardize incoming data formats so mapping rules behave predictably. Examples include phone formats, country codes, name casing conventions, and standardizing picklist values.

If you do not normalize, you may map “Qualified - High” in one system and “Qualified - High ” in another with trailing spaces. The connector might treat them as distinct, leading to missing picklist mapping. Even if the connector handles it, reporting can become fragmented.

Normalization does not have to be perfect. It just needs to be consistent, and it needs to be measurable.

Picklists and historical values

Legacy systems often allow values that the new CRM does not. Or the new CRM has different picklist labels.

For historical records, you face a fork in the road:

- Map the legacy value to the closest new value, even if it loses some nuance
- Store the legacy value in an “original value” field for reference, and map to a neutral category that keeps the record valid

In my experience, the second option prevents data loss. It also avoids blocking migration when the new CRM enforces strict picklists.

But it introduces reporting complexity: you may need to build reports that use the mapped value for current insights and refer to “original value” when investigating old records.

Again, the intent matters. If the business cares about historical accuracy for compliance or audits, preserving the original values is worth the extra complexity.

Relationships and referential integrity

Mapping is also about relationships: accounts to contacts, opportunities to accounts, cases to contacts, activities to who they happened with.

A migration that populates objects correctly but attaches relationships incorrectly will look superficially fine until a user tries to find context. People can tolerate missing fields longer than they can tolerate a broken timeline.

Referential integrity issues often come from ordering. If your migration loads activities before contacts exist in the new CRM, the migration may skip or temporarily store activity links.

Some migration tools support upserts with deferred resolution. Others do not. If yours does not, you must control load order carefully:

- Create canonical parent records first
- Then load children
- Then resolve cross-references

I typically run a small set of records that represent the densest relationships, not the easiest ones. A contact with multiple opportunities, multiple activities, and multiple cases will expose mapping and ordering problems quickly.

Ownership, assignment, and permissions

Ownership in CRMs is more than a name. It affects what a user can see, what queues process, and who receives notifications.

During migration, owners might be missing if the new CRM has different user identifiers, if roles do not match, or if some old users are not present. If you do not map ownership carefully, you might end up with default owners that mask the original responsibility structure.

This is one place where “good enough” gets dangerous. Users may believe data is wrong because it landed under the wrong owner. Leaders might believe forecasts are off because assigned opportunities moved between teams.

A robust plan includes:

- A user mapping table between old user IDs and new user IDs

- A clear default ownership strategy for unmapped users
- A validation report that shows how many records fell back to defaults

If defaults are inevitable, document them and validate the fallout. Otherwise you will have a mystery that only shows up after teams start using the CRM.

Testing strategy: validation that reflects how people actually use the CRM

Testing migrations is not just about data counts. It is about user workflows.

The easiest tests check that records exist. The better tests check that records behave like records.

I like to build test scenarios based on real tasks:

- A sales rep opens an opportunity and expects to see call notes, emails, and the correct account context.
- A manager checks pipeline by stage and expects historical stages to align with the legacy reporting logic.
- A support agent reviews a case and expects attachments and communication history.

To keep it practical, choose a small set of accounts and contacts that cover edge cases, like merged duplicates, picklist differences, missing owners, and time zone sensitive timestamps.

A short, brutal reconciliation mindset

At minimum, validate counts and reconciliations at three levels:

1. Total record counts for each object type, by time window.
2. Record counts after deduplication, with an explanation for changes.
3. Activity and relationship counts, with spot checks on the records that changed most.

If you can explain every discrepancy with either a rule change or a known data issue, the migration feels controlled rather than chaotic.

Operationalizing the migration: roles, approvals, and change control

Even the best plan collapses when teams make ad hoc decisions mid-migration.

You need a decision process for the hard cases: duplicates that do not match confidently, picklist values that have no mapping, and relationships that conflict.

In my projects, I establish two layers of review: technical validation for schema and data integrity, and business validation for deduplication and mapping decisions that change interpretation.

When changes happen late, they should be logged like code changes. Otherwise you end up with two environments that were supposed to be identical, but one has a different mapping rule for “status” or a different canonical key for accounts.

That is how silent divergence happens.

Data governance after migration: the work does not stop at go-live

Once the migration completes, duplicates do not stop. Data entry keeps happening, integrations keep writing, and users still import spreadsheets.

To prevent reintroducing duplicates, your migration strategy needs to seed governance into the new CRM.

That includes:

- Deduplication rules that align with the canonical key logic used in migration
- Field validation that reduces input ambiguity (for example, enforcing country codes or email formatting)
- Integration constraints that prevent multiple systems from creating new records when an existing one should be updated

The goal is not to eliminate duplicates forever. The goal is to make duplicates rarer, easier to detect, and less damaging when they appear.

A pragmatic checklist for history, duplicates, and mapping

If you only remember one thing from all of this, remember that history, duplicates, and mapping are one system. Fixing one without the others often creates new problems.

Here is the checklist I use to keep the work aligned. It is intentionally short because the real details live in the validation artifacts.

- Confirm what “history” includes for each object: activities, field changes, notes, attachments, and audit-like events.
- Define a canonical key strategy for duplicates, with confidence tiers and a merge approach that preserves timeline readability.
- Normalize incoming values before mapping, especially phone, country, and picklist labels.
- Validate relationship integrity by load order and run spot checks on the most interconnected records.
- Reconcile counts and investigate discrepancies with documented rule changes, not assumptions.

Common edge cases that deserve real attention

Every migration has the same categories of weirdness. The difference is how frequently they show up and how costly they become.

The “same email, different people” scenario

Email is often the best key, until it is not. Shared inboxes, aliases, junior staff using a mentor’s email format, or company-wide addresses can cause false matches.

When this happens, you need a rule that considers additional attributes. Examples include matching on name plus country, or requiring agreement on account domain for certain confidence levels. If you cannot reliably disambiguate, quarantine those candidates for review rather than merging automatically.

The “different email, same person” scenario

People change roles, and their email changes. If you deduplicate solely on current email, you will split history across multiple contact records. That can make old activity look like it belongs to someone else, which damages reporting and customer continuity.

If the legacy system stored any stable identifier or an email history field, migrate it. If not, consider capturing older emails found in activities or notes into an alternate email field or a related table. Even a simple “previous email” field can enable safer deduplication later.

Stage transitions that do not map cleanly

Opportunities and cases often use stage logic that differs across systems. The new CRM might have a different stage taxonomy, or it might treat closed stages differently.

When mapping stages, you need to consider not just the current stage, but the historical stage values. Users may care about how the opportunity progressed, not only where it ended.

If you have to map many legacy stages into fewer new stages, decide whether the migration should preserve the mapped stage for reporting, or preserve the original stage for auditing and show mapped stage only for operational reporting. Either is valid, but mixing them without a plan creates confusing timelines.

Deleted or inactive records

Some legacy systems treat soft deletions as state. Others keep historical records accessible but mark them inactive.

During migration, you need a clear policy on what to do with inactive or deleted records:

- Keep them for audit and history views
- Exclude them to reduce clutter
- Keep them but mark them in a way that prevents user workflows from treating them as current

This interacts with duplicates. If one duplicate is active and another is inactive, merge rules need to define whether history from inactive records moves into the active canonical record, and whether the inactive record should remain for audit trail.

Rows with missing keys

It happens constantly: a contact row with no email, no phone, and an incomplete name. A lead row with only a company and a vague timestamp.

If your migration requires a canonical key, those records either fail validation or get mapped into a “misc” bucket. Neither outcome is great.

You need a strategy for missing keys:

- Attempt to reconstruct keys from context where possible (for example, matching by external IDs in activities)
- Preserve such records but flag them for cleansing after migration
- Use a default placeholder that still allows relationship mapping without creating false merges

This is another area where review matters. If a record is missing keys, it is likely also missing context, and crm automation duplicates decisions become more dangerous.

What “done” looks like for a migration team

A migration is not done when the data loads.

It is done when you can take a user through a realistic workflow, on realistic records, and the experience feels consistent with what they expect. That means timeline order makes sense, duplicates are not harming attribution,

and reporting at least holds together.

If your validation artifacts tell a coherent story, you can defend the outcome. If they do not, you can spend weeks chasing complaints that trace back to a handful of mapping decisions made without enough context.

The best migrations feel boring on go-live day. People work. Managers get answers. Support teams see the history they need. Behind the scenes, the migration team has already done the hard part: converting messy legacy meaning into something the new system can reliably carry.

If you are planning a migration right now, the fastest path to reducing risk is to tighten the “meaning” layer: define history scope, formalize duplicate confidence rules, and map not only fields but behavior. That is where the real payoff lives.