

A Great LIMS Can't Beat Bad Data

Welcome to InCogSights™, Issue Three.

Every LIMS project I've worked on, going back to Barr Labs in 1995, gets scoped the same way. The vendor demo looks great. The workflows map cleanly. The timeline gets built around configuration, validation, training, go-live. And buried in that timeline is a single line item: "data migration." One line, a few weeks, done.

It's never a few weeks. It's the part of the project that's mostly invisible until it isn't, and by the time it surfaces, it's already eaten the budget and the schedule.

That's because migration isn't really a task inside the project. It's the foundation the whole project gets built on. Every workflow, every report, every audit trail, every piece of trending and analysis you'll ever pull from the new LIMS depends on the data sitting underneath it. A great LIMS on a shaky foundation of bad data is still a shaky foundation, no matter how well the system itself is configured. Get the foundation right and everything built on top of it holds. Get it wrong and you'll be patching cracks for years to come.

Before getting into what actually blows up that foundation, it's worth asking a question most projects skip entirely: should all of this data even be migrated in the first place?

Sometimes the answer is no

Not every data set is worth carrying forward. There's a point in some projects where migrating years of data built around an old workflow doesn't just cost time. It locks in the very structure you're trying to leave behind. Starting with a clean subset of that data, built to match the new system's to-be workflow rather than the old one's habits, can feel like a step backward when the timeline is already tight. In practice it's often the faster path, because you're not troubleshooting a foundation that was never designed for where the lab is headed. The dividends show up later: fewer workarounds, less reporting that needs a translation layer, and a system that behaves the way it was configured to instead of the way the old one happened to work.

That decision, migrate or rebuild clean, has to be made on purpose. Left to default, everything gets carried forward because nobody stopped to ask. For whatever data does get migrated, here's what I've seen blow up the timeline and the budget most consistently.

It's never in one place

Nobody has clean, centralized data waiting to be moved. It's spread across the outgoing LIMS, a couple of departmental spreadsheets nobody admitted to keeping, an Access database from 2011, and a filing cabinet of paper records that predate all of it. Every one of those sources needs its own extraction approach. And even where the same record genuinely exists in more than one place, it rarely matches character for character: "Newton, NY" in one system and "Newton,NY" in another look identical to a person and completely different to a matching algorithm doing string comparison, so the same client, site, or supplier ends up counted twice instead of reconciled into one. Untangling that is where the real time goes, not the import itself.

Garbage in, garbage live

Data that's been "good enough" for years in the old system rarely survives contact with a new one. Duplicate records, inconsistent naming, results logged in three different units for the same test, specs that were updated in practice but never in the record. None of that gets fixed by the migration tool. If it's not cleaned before handoff, it goes in dirty and it comes out live, which is a much more expensive place to find it.

"This is how we've always done it"

Two things tend to happen at once, and they compound each other. Nobody goes back and asks whether the test setups and recorded fields still make sense, so obsolete or inefficient structures get carried forward by default. And when someone does raise the question, the instinct is to preserve the old process exactly rather than let the new system do it better. A migration is one of the only times you'll ever have a real excuse to fix long-standing workflow debt. Reproducing the old system faithfully in a new platform gets you a new system with the same problems.

The field doesn't care what you meant

This is the least glamorous failure point and one of the most common, because it shows up in ways that look trivial until you count the damage. Case sensitivity is a classic: a picklist value of "Assay" in the source system is not the same string as "ASSAY" or "assay" to a system doing an exact match, even though they mean identical things to the chemist who entered it. If legacy data was entered with three different casing conventions over the years and the target's master list expects one, every import that relies on an exact match either fails outright or, worse, silently creates duplicate lookup entries that fragment your reporting for years afterward. The same category of problem shows up elsewhere: a text field that allowed 500 characters in the old system gets truncated at 100 in the new one, a numeric field expects a number and gets "TBD" or "see notes," dates arrive in three formats from three sources. None of this is exotic, it's just detail work, and detail work is exactly what gets skipped when a timeline is already tight.

Validation is not a checkbox

The most expensive mistake is a shallow check. Spot-checking twenty records and calling the migration validated feels efficient, but it means the gaps get found by an analyst mid-run, six weeks after go-live, in a regulated environment where that record now matters. A real reconciliation (record counts, field-level comparisons, statistical sampling, at minimum) costs time up front. It's cheaper than the alternative every time.

A great LIMS can't beat bad data. It can only inherit it. The labs that come out ahead are the ones that clean house before the system ever sees the data, even when that means leaving some of it behind entirely.

If you're planning a migration and want a second set of eyes on it before the data moves, reach out.

Thanks for reading, and I'll see you in the next issue.

Matt