

When systems and data stop agreeing

Nobody decides that two systems should disagree. They drift, one sensible fix at a time, until the figures in a meeting no longer match and nobody can say which one is right.

Author	Amit Govil, Founder and CEO, XONIK
--------	------------------------------------

Published	October 22, 2026
-----------	------------------

Reading time	30 min read
--------------	-------------

Online	https://xonik.com/insights/reports/when-systems-and-data-stop-agreeing/
--------	---

IN BRIEF

Systems stop agreeing because the same fact is entered or changed in more than one place, definitions drift, integrations fail without a signal and no system is named as the record. People then reconcile by hand. What changes it is one named source per record and per definition, loud failures, scheduled reconciliation with an owner, and a retired workaround.

KEY FINDINGS

1. The same fact enters through several doors, and each door makes its own copy

Each entry point is reasonable alone, because each team needs the record now. A correction is complete only when it reaches every copy, and an estate that cannot list its copies cannot say that it has.

2. The same word means different things in different systems, and each meaning is correct by its own rule

The trouble starts when a report adds them together. The answer is not to force one meaning on every user but to give each meaning its own name, a written rule and an owner, and to make every report say which it uses.

3. Integrations fail silently because skipping a bad record is often a setting someone chose

A job can finish while records are left behind, a success response can mean only that a message was received, and retries create duplicates. The gap is found far from its cause, in a report, a complaint or an audit.

4. No system is named as the system of record, so every system is partly right

Ask in which system a correction must be made for it to hold. If the answer is that it depends, there is none for that fact. A definition, a system of record and an interface are three layers and should be kept apart.

5. The reconciling spreadsheet becomes the real system, and each quick fix adds another interface

The spreadsheet holds rules the estate lacks and a figure that exists nowhere else, and an adjustment made downstream leaves the source unchanged. An interface with no owner works until the first

change.

6. The cost arrives as contested decisions, and reporting or AI built on the estate inherits the disagreement

A model does not settle a disagreement between two systems. It reads one or both and answers with the same confidence. An estate may be good enough for a monthly summary and not for an automated decision.

What does it mean when systems and data stop agreeing, and why does it matter now?

This report asks why the systems in a business come to hold different answers to the same question, and what keeps them together. It is written for the leader who lives with the consequence: a finance chief who cannot close the books on one set of figures, an operations head whose teams quote different order status, a technology chief asked why every change touches so much. It is also for the architects and data leads asked to put it right.

Take a customer. The shop knows an account, billing knows a payer, support knows the person who writes in, and marketing knows an address on a list. Each system was bought or built to do one job, and each needed its own version of the customer to do it. Nothing in that is a mistake. The trouble starts when the versions are asked to be the same thing, in a board report, an audit, a call with a customer or a model that learns from them, and they are not.

A few terms are used exactly in what follows. A record is the set of facts about one thing, such as a customer or an order. A copy is any place that holds those facts, from a database table to a spreadsheet. A system of record is the one place accountable for the correct state of a record. The European Interoperability Framework of 2017 describes the same idea for public administrations as an authoritative source, with one organization responsible and accountable for collecting, using, updating and preserving the information (section 4.3.3). Drift is what happens when copies change at different times, by different routes and under different definitions.

The question is pressing because a new tool is easy to adopt and a connector is a menu item, so a new copy of the customer can exist by the end of an afternoon with nobody accountable for it afterwards. And the work leaders now ask for, reporting they can rely on and AI that reads company data, takes the estate's disagreements as its raw material. The UK Government Data Quality Framework (2020) ties public services and the country's AI ambitions to data that is fit for its intended purpose. NIST's AI Risk Management Framework (2023) lists, among the ways AI risk differs from conventional software, data that may not be a true or appropriate representation of the context of use and datasets that go stale against the setting where they are deployed. A model does not settle a disagreement between two systems. It reads one or both, and answers with the same confidence.

Method and limits

This is practitioner analysis. It rests on three things. The first is our delivery experience since 2006 in integration, data and architecture work, described as pattern and mechanism. Every engagement is under NDA, so no client, case, count or share appears, and nothing here says how often any pattern occurs. The second is public documents, read for the passages cited: data quality and interoperability frameworks from

the UK government, the European Union and NIST, guidance from the UK Information Commissioner's Office, and the documentation of Stripe, Amazon Web Services, Microsoft and Google for what their own services do. The sources section gives versions and the date each was read. The third is reasoning from how copies, schedules and retries work, and where a point is reasoning we say so.

Public evidence on this subject is thin. Official work on data quality is mature on definitions and on what to measure, and it is written mostly for statistics offices and public bodies. Vendor documentation shows what a service does, not how often anyone is caught out by it. So the report claims mechanism, and each mechanism can be looked for in an estate in an afternoon.

- It does not say how common drift is, how fast it develops or what it costs in money. No source read measures those things, so none is stated.
- The public frameworks are written for statistical and public-sector data. We use them for definitions and for principles that carry across, and we do not claim that a business must meet them.
- Vendor documentation describes a service on the date it was read. Services change, and a reader should check current documentation before relying on a detail.
- The set of mechanisms is a working classification, not a standard. Real estates mix them, and one incident often involves several.
- Reconciliation shows that two systems differ. It does not decide which is right, and that decision belongs to the business.
- Nothing is said about a specific product beyond what its own documentation says.

Why does the same fact end up with more than one entry point?

A fact has a single entry point only when someone designs it that way. Otherwise entry points accumulate. The sign-up form creates a customer. Sales creates one when a deal opens. Support creates one when an unknown person writes in. A partner's file arrives as a batch. A new system comes with a migration that loads whatever the old one held, duplicates included. Each entry point is reasonable alone, because each team needs the record to do its work now and cannot wait for another team's system to learn about the customer.

Two things follow. The same entity is created more than once, so the estate holds several records for one customer. And the same record is changed from more than one place, so two edits cross. The UK Government Data Quality Framework lists both among the problems that arise when data is collected: errors in manual entry, duplicated data, inconsistent formats. It makes uniqueness a core dimension, meaning one record for each entity, and its own example is a school database that holds one student under two forms of the same name.

What you can see: two records for one customer with different contact details, both edited recently. A support agent who updates one while billing updates the other. A rule applied from memory, such as always searching before creating. A customer list that grows on every import and never shrinks.

What it costs: which record survives is settled by whoever finds the pair, and the answer is rarely written down. A merge in one system does not reach the others. A customer who asked to be removed from a list can remain on another. The UK Information Commissioner's guidance on the right to rectification asks that an organization have procedures to inform recipients when it corrects data it has shared with them. That concerns other organizations, but the principle carries to copies inside an estate: a correction is complete

only when it reaches every copy, and an estate that cannot list its copies cannot say that it has.

The first piece of work is therefore a list of entry points for each fact that matters, not a list of systems. Exhibit 1 shows one fact, a customer's status, and the copies it breaks into, with the reconciling spreadsheet as the last and least visible of them.

[The Government Data Quality Framework \(GOV.UK, 2020\)](#)

[Right to rectification \(Information Commissioner's Office\)](#)

EXHIBIT 1

One fact breaks into its copies



Each copy is correct at the moment its own event last fired. The spreadsheet is the copy that nobody designed and that everyone reads.

Why do the same words come to mean different things?

A system stores a word as a rule. Active in billing means a subscription with a payment method that has not been stopped. Active in the product means the person signed in recently. Active in marketing means someone who has not unsubscribed. Closed means signed for a sales deal, resolved for a support case, and no further postings for a ledger period. Revenue can mean what was invoiced, what was paid, what was recognized under an accounting policy or what was booked. None of these is wrong, because each serves its owner. They become wrong when a report adds them together or compares them as if they were one.

Definitions also change inside one system. A product team changes what counts as a recent sign-in. Finance changes a recognition rule. The change is correct where it is made and invisible everywhere else, so a trend line acquires a break that has nothing to do with the business. The European Statistics Code of Practice, adopted in 2017, treats this as a duty of statistical producers. Statistics are to be compiled on common standards for scope, definitions, units and classifications across sources (indicator 14.3), and are to be comparable over a reasonable period (indicator 14.2). Our reading is that a change of definition is a recorded, dated decision and not a quiet edit.

What you can see: two reports with the same label and different figures, each reproducible and each correct by its own rule. An argument that ends when someone asks how a number is calculated and nobody can

say. A dashboard filter named for a business term whose rule lives in a query nobody reads.

The UK framework defines consistency as the extent to which a value fails to contradict the values held for the same entity, in one data set or across several. A difference of definition produces exactly this: values that contradict because they answer different questions. The framework also says that quality is fitness for purpose and that users' needs may conflict. That points to the response: do not force one meaning on every user. Give each meaning its own name, written rule and owner, and make every report say which it uses.

What it costs: the argument moves from the business to the analysts. A leader asks for revenue and receives several figures, each defended, and time that should go to the decision goes to establishing what was asked.

Why do integrations fail without anyone noticing?

An integration is a copy made on a schedule or on an event, and it is usually built to carry the normal case. The cases it meets later include a record that fails a validation rule, a record that refers to something not yet there, a system that is down at the moment of the run, and a sender that changes its format. What the integration does in each case is a decision, whether or not anyone made it deliberately. Often the decision is to skip the record and carry on, because stopping everything for one bad row looked worse.

Vendor documentation shows that this is an explicit setting. Microsoft's documentation for the Azure Data Factory copy activity describes two ways to handle a failure: abort the copy, or enable fault tolerance and continue, skipping the incompatible data. Skipped items are recorded only if logging is configured. Google's BigQuery documentation for loading CSV files sets the number of bad records a load may ignore at none by default, and describes the option to raise it, in which case the load proceeds without them. These are reasonable features. The point is that a pipeline can be set, for good reason at the time, to finish while records are left behind, and whether anyone sees the gap depends on whether the skipped records are counted and reported to a person.

A second quiet failure is the difference between receiving and doing. Stripe's webhook documentation tells a receiving system to return a success response quickly, before any complex logic, and gives the example of returning it before marking an invoice paid in an accounting system. That is sound engineering, because it stops the sender timing out. It also means the success the sender sees is an acknowledgment of receipt, and whether the invoice was then marked paid is known only to the receiver. That step is our inference from the documentation.

Timing and order are a third group. Stripe states that it does not guarantee delivery of events in the order they were generated, and advises that an endpoint should not depend on order. Amazon's documentation for standard queues in Amazon SQS says that delivery is at least once, that more than one copy of a message might be delivered, and that messages may occasionally arrive out of order. A retry exists to prevent loss, and it creates a duplicate unless the receiver recognizes the repeat. A receiver that treats every message as new applies a change twice. One that assumes order can apply an older change over a newer one, so that a cancel message arrives before the order it cancels and the record ends up open.

What you can see: record counts that differ between two systems, found by a person and found late. Records with different values in the same field, where the later job won because it ran later and not because it was right. Integrations whose only monitoring is that the job did not crash. In our delivery work the silent kind does the longest damage, because it leaves no failure to investigate. The evidence is an absence, and

an absence is found only when someone looks for what is missing.

What it costs is time, because the first discovery is made far from the cause. The gap shows up in a report, a complaint or an audit, the trail back through jobs and mappings is long, and by then the error has been copied onward. The European Statistics Code of Practice asks that source data, integrated data, intermediate results and outputs are all regularly assessed and validated (indicator 12.1): each stage, not only the last. Exhibit 2 follows one order through an estate and marks the point where the copies stop agreeing.

[Receive Stripe events in your webhook endpoint \(Stripe\)](#)

[Amazon SQS standard queues \(Amazon Web Services\)](#)

[Fault tolerance of copy activity \(Microsoft Learn\)](#)

[Load CSV data from Cloud Storage \(Google Cloud\)](#)

EXHIBIT 2

Where one order's copies stop agreeing



The first failure is not at the report. It is at the hand-off, and the status shown there says the job succeeded.

Why does no system end up as the system of record?

Nobody decides against having a system of record. It goes unnamed. When the second system arrives, it needs the customer, so it asks for a copy. When the third arrives, it copies from whichever system the team knows best. Over time each system is the source for some fields and a copy for others, and the map of which is which exists only as an understanding among the people who built the integrations. Every system is partly right.

The practical test is simple. Pick a fact, such as a customer's billing address, and ask in which system a correction must be made for it to hold. If the answer is that it depends, or all of them, there is no system of record for that fact. If two systems both accept the edit and both send it on to each other, the estate holds two records that overwrite one another, and the outcome depends on which job ran last.

The European Interoperability Framework describes the alternative for public administrations. A base registry is a trusted and authoritative source of information, with one organization responsible and accountable for collecting, using, updating and preserving it. Where a registry is distributed, a single entity must be accountable for every part and one for coordinating the parts (section 4.3.3). The framework recommends an information management strategy at the highest possible level to avoid fragmentation and duplication (recommendation 31) and the publication of the documentation others need to connect to an authoritative source (recommendation 38). The structure carries over: an owner for each record, a declared

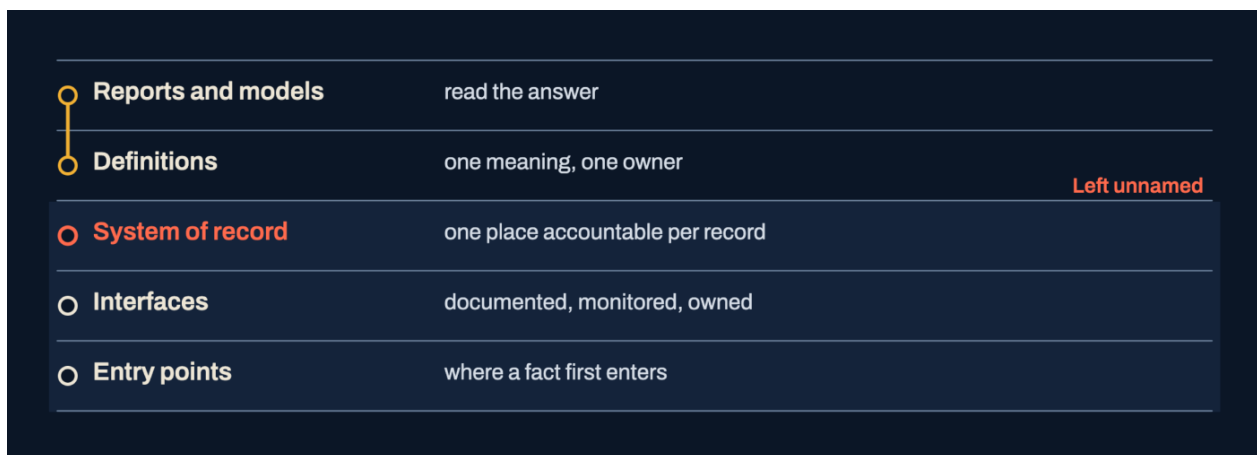
source and a published way to read from it.

Three things are easily run together and should be kept apart. A definition says what a record means. A system of record says where its correct state is kept. An interface says how other systems reach it. Where the three are not separated, a change to one is a change to all. Exhibit 3 puts them in layers and marks the layer we look for first: the declaration that makes one system accountable.

The cost shows when anything changes. A new regulation, a new product line or a migration needs an answer to where something lives, and the answer is a research project. A retired system takes with it the only copy of something nobody listed.

EXHIBIT 3

Definition, system of record and interface are three layers



A definition says what a record means, the system of record says where its correct state is kept, and an interface says how others reach it. Without the middle layer, the layers above it rest on a guess.

Why does the reconciling spreadsheet become the real system?

When two figures disagree and a decision is due, someone reconciles them. The first time it is a favor, done in a spreadsheet: pull both reports, match on a key, list the differences, adjust. It works, so it is repeated at the next review, and then every cycle. The spreadsheet gains a tab for exceptions, a column that marks which system to believe, a macro, a note about a customer who is always wrong in billing. It is now doing a job the systems were meant to do, which is to hold the rules for how the estate's records relate, and the person who maintains it is the only one who can say what they are.

The UK framework's foreword describes the pattern in the public sector. It cites a 2019 report of the Public Accounts Committee, which showed that data had not been treated as an asset and that it had become normal to work around poor-quality, disorganized data. The framework's guidance on root causes puts the cost plainly: problems become accepted, repeated fixes become part of the process, and the cost of not fixing can be underestimated as people come to accept several problems at once.

What you can see: a file with a person's name in its title. A figure in a board pack footnoted as adjusted. A plan to automate the reconciliation that stalls because nobody can write down the rules. When the owner is away the number is late or wrong, and nobody can tell which.

The spreadsheet is also a copy. It holds the adjusted figure, a version of the fact that exists nowhere else, and that figure feeds onward into a forecast or a covenant report. The adjustment is not made at the source, so the next report built from the source starts again with the original error. The UK guidance asks that problems be fixed as close to the source as possible, and it lists correcting low-quality data directly as an action that can be risky and should come after the other fixes.

The way out is not to ban the spreadsheet. It is to treat it as a specification. Its tabs and columns are the rules the estate lacks, written down in the one place that was ever trusted. Moving them to the source, or into an automated check with an owner, retires the workaround and keeps the knowledge.

The reconciling spreadsheet is a specification: its tabs and columns are the rules the estate lacks, written down in the one place that was ever trusted.

Why does every fix add another interface?

When two systems disagree, the quickest fix adds a connection: a nightly sync to bring billing in line with the shop, a middle tool that translates between two formats. Each fix is a new copy with a schedule, so it brings the whole set of problems above: a new entry point, a new mapping that holds a definition, a new place to fail quietly. The possible connections between systems rise faster than the number of systems, because every pair can be joined, and an estate joined pair by pair has no single place from which to see how a record travels.

The order of operations becomes unclear. If the shop updates billing, billing updates the customer system and the customer system updates the shop, a change travels in a circle, and the last write wins on grounds nobody chose. And the interfaces have no owners: the team that built the sync has moved on, the vendor changes a field, and the sync keeps running against it, partly. The interoperability framework recommends publishing the documentation needed to connect to an authoritative source (recommendation 38). An interface that is not documented and owned works until the first change.

What you can see: an integration diagram nobody can draw from memory and that differs with whoever draws it. Connectors from a vendor's catalog switched on by teams who did not tell anyone. A middleware layer that holds business rules in its mappings.

What it costs: every change takes more coordination than the last. When a team needs meetings with several others to change a field, that field is a shared interface nobody manages as one. The effect compounds the pattern described in the perspective on why each change costs more than the one before, because every interface added is one more thing that every later change has to check.

Treating an interface as a product changes this: an owner, a documented contract, a version, monitoring that counts what went in against what arrived, and a retirement date.

What does the disagreement cost the business?

The cost does not arrive as a line item. It arrives as consequences, and a leader will recognize each of them.

Decisions are taken on contested figures. A price change, a hiring plan or a covenant report goes ahead on a number someone has adjusted, or on the number from the system the decider happens to trust. The decision may be right. Its basis cannot be reproduced, and the next person to ask will be given a different

answer.

Meetings are spent on whose number is right. The first part of the review becomes reconciliation, the review shifts from the decision to the data, and the people who prepare it learn to bring a defense with every figure.

Change needs more coordination each time because the rules are unrecorded. Small changes are postponed since the cost of checking exceeds the benefit, and the estate ages.

Confidence drops and stays down. People stop trusting a figure that has been wrong once, even when it is right again. The UK framework says that poor or unknown quality data weakens evidence and undermines trust.

Customers and regulators see the gap. A customer is told two things. A request to correct a record misses a copy. An audit asks for the lineage of a figure and nobody can draw it.

Reporting and AI are built on top of the disagreement. A report that joins two systems under different definitions produces a figure with false precision. A model fed records from several systems learns or repeats the disagreement. The UK framework's central idea helps here: quality is fitness for purpose, and the level required varies with the purpose. An estate may be good enough for a monthly summary and not for an automated decision, because the summary absorbs a disagreement that the automated step cannot.

Exhibit 4 sets the symptoms a leader sees against the mechanism behind each, so the pattern can be read from the symptom backward.

EXHIBIT 4

Symptoms and the mechanism behind each

What you see	The mechanism behind it	Where to look first
Two reports carry the same label and give different figures	Definition drift: each report applies its owner's rule	The written rule behind each figure
A customer is told different things by two teams	Several copies updated at different times, with no declared record	The system in which a correction must be made for it to hold
Record counts differ after a job reports success	A silent failure: skipped or rejected rows, or a partial run	What the job skipped and rejected in its recent runs
A record reverts to an old value	Out-of-order delivery, or two systems overwriting each other	Which system accepts the edit and which one sends it on
The same change is applied twice	A retry delivered a duplicate and the receiver did not recognize it	How the receiver recognizes an event it has already handled
A spreadsheet adjusts the official figure	A manual workaround holding rules that no system holds	The spreadsheet's tabs and columns, read as a specification
A small change needs several teams	Interfaces added pair by pair, with no owners	Every interface that touches the field, and who owns each
A model gives confident answers that differ	Inputs from copies that disagree	The sources the model reads, and the record for each fact

Read from the left column to the middle one. The symptom is rarely where the cause is, so the third column says where to start.

What changes the outcome?

The measures that change the outcome follow from the mechanisms, and they work as a set. Adopting one exposes the need for the next.

Name one source of truth for each record. One system is accountable for its correct state, and every other system either reads from it or states that it holds a copy. Write it down with a named owner, so the declaration survives changes of staff. The interoperability framework's wording is a good standard: one party responsible and accountable for collecting, using, updating and preserving the information.

Name one definition for each term, and name different things differently. Where the business needs two meanings of active, they get two names, each with a written rule and an owner. Where a rule changes, the change is a recorded and dated decision with its effect on trends stated.

Make failures loud. A job that finishes while records are left behind must say so to a named person. That means counting what went in against what arrived and what was rejected, treating a skip as an event, and never setting a tolerance for bad records without an alert for them. It means receivers that recognize a repeated message and do not depend on order, as the Stripe and Amazon documentation each ask of the builder.

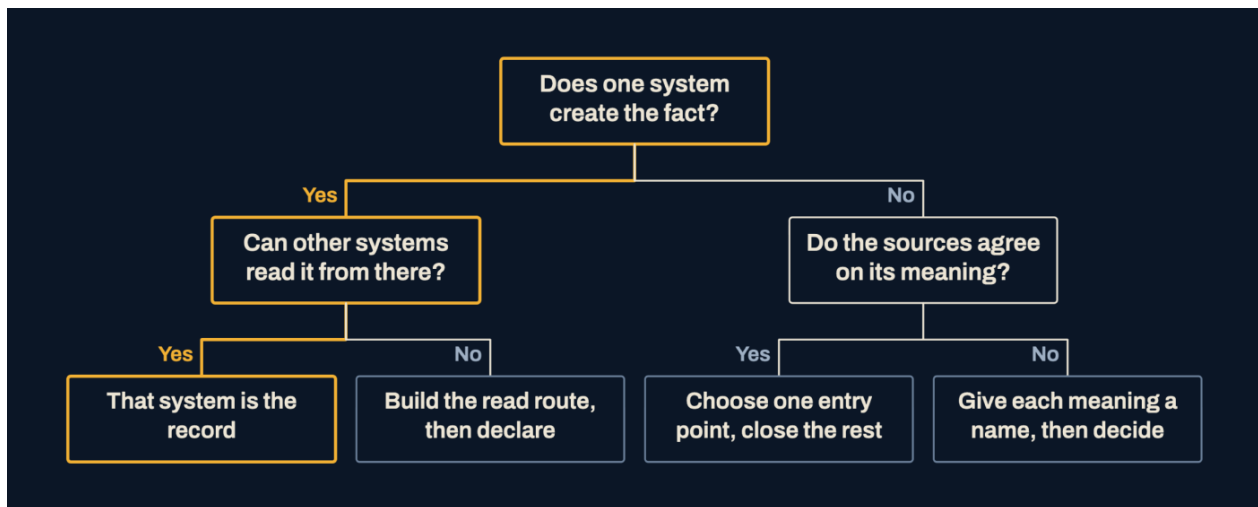
Reconcile on a schedule, with an owner. Compare the two ends of a hand-off record by record, at a frequency matched to how fast the fact changes, and put a person's name on the result. The Code of Practice asks that statistics from different sources be compared and reconciled (indicator 14.4).

Retire the workaround, and design interfaces as products, each with an owner, a contract, a version, monitoring and an end date.

The decision on which the rest depend is the choice of system of record. Exhibit 5 sets out the questions we ask, in order: whether one system creates the fact, whether others can read it from there, and, where no single system creates it, whether the sources agree on what it means. That last question matters because a disagreement of meaning cannot be settled by choosing a source. The two things must be named differently first.

EXHIBIT 5

How to choose the system of record



The last branch is the one to take care over. A disagreement about meaning cannot be settled by choosing a source.

How do you resolve a disagreement once you have found one?

Finding a disagreement is the easier half. The harder half is choosing what to do about it, because the right action depends on its kind. Treating a difference of definition with a merge, or a duplicate with a rename, leaves the cause in place and the symptom returns. The UK guidance on root causes is the frame: establish the cause before choosing the fix, and fix as close to the source as possible.

Exhibit 6 lists the ways to resolve a disagreement and when each fits. They are not alternatives to be ranked. A single estate will use several, in an order that follows the cause.

EXHIBIT 6

Ways to resolve a disagreement, and when each fits

Way to resolve	What it does	When it fits, and what to watch
Declare one record and read from it	Other systems stop editing the fact and read it from the record	The fact has a clear origin. Watch the read route: a copy kept for speed must say it is a copy
Close extra entry points	One form, one import path, one owner; the others become links	The same fact is typed in several places. Watch for the shortcut people lose and give them a reason to accept it
Rename the terms	Each meaning gets its own name, written rule and owner	The disagreement is about definition and not data. Watch that every report says which term it uses
Merge duplicate records	Records for one entity are combined under one key	The estate holds several records for one entity. Watch that the merge reaches every copy, and keep a trail

Reconcile on a schedule	Both ends are compared and a named owner is alerted	Two systems must each hold the fact, for speed or independence. Watch that it finds differences and does not say who is right
Retire a system or an interface	A copy or sync with no purpose left is switched off	A system repeats another's job, or a sync only repairs another sync. Watch what depends on it, and set the date
Accept a declared difference	Two figures are documented as different by design	The purposes differ and each figure is right for its use. Watch both names and review when the use changes

The cause decides the way. A merge does not fix a definition, and a rename does not fix a failed job.

WHERE IT GOES WRONG

The cause stays in place

Treating a difference of definition with a merge, or a duplicate with a rename, leaves the cause in place and the symptom returns.

What does reconcile and fix look like in practice?

Reconciliation is not a project. It is a loop with an owner. The owner compares the two ends of a hand-off on a schedule: the record counts, the set of keys on each side, and the values of the fields that matter. A difference is a finding, and the owner traces it to its cause: a definition, an entry point, a failed job or a retry. The fix is made at the source, so that it holds for everyone. The rule the fix established, such as a canceled order cannot be reopened by a late update, is recorded and where possible turned into a check, so that the next run tests it. Then the loop runs again.

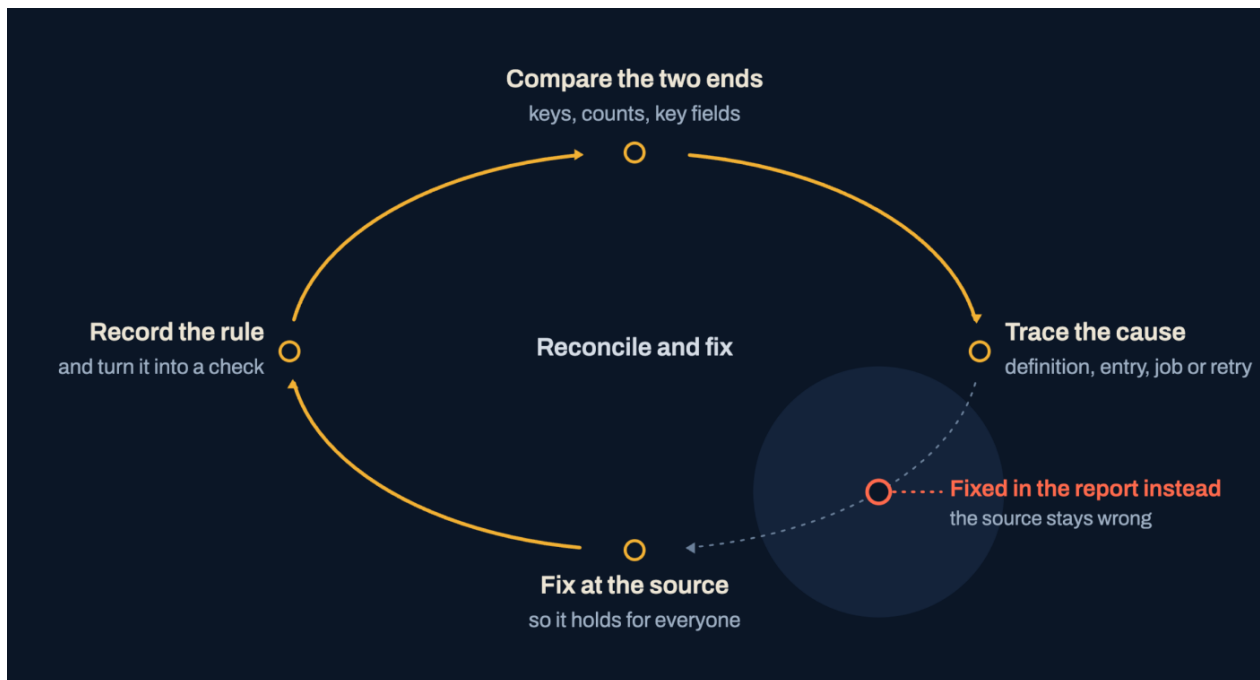
Two design points decide whether it works. A comparison needs a key that both sides share, or a mapping table someone maintains, because without one matching becomes judgment and the loop slows to the speed of the person doing it. And the tolerance needs a name. Timing makes some differences legitimate, since a record may be in transit, so the check states how long a difference may exist before it is a finding. The UK framework's timeliness example, a change entered within an agreed time of collection, is the form to copy: a stated limit that the data is measured against.

The cadence follows the cost of being wrong: often for a fact that drives money or a promise to a customer, before each report for a figure reported once in a period. The owner is a person with a role and the authority to get a source fixed, not a team alias nobody answers for.

Exhibit 7 shows the loop and the place where it most easily stops: the fix made downstream, in the report or the copy, so that the source is unchanged and the same difference returns at the next run.

EXHIBIT 7

The reconcile-and-fix loop



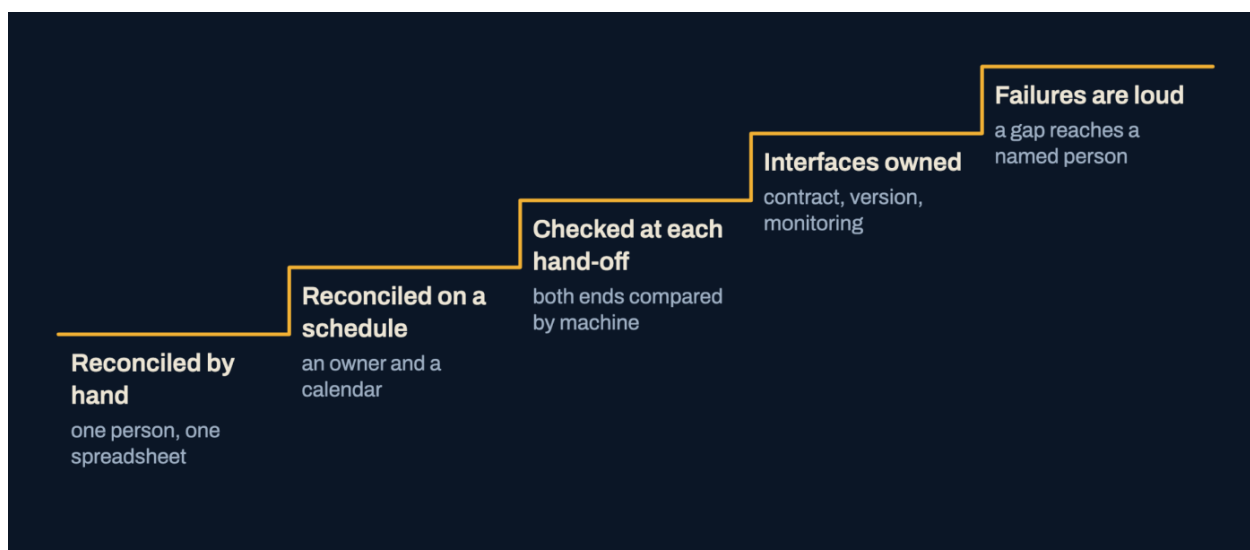
The loop only converges when the fix lands at the source. A fix in the report removes the symptom for one reader and leaves it for the next.

How do you know it is getting better?

A ladder helps a leader ask for evidence without reading code. Exhibit 8 sets out the levels we use, from reconciliation done by hand by one person to an estate in which a failure reaches a named person without anyone looking for it. The UK guidance introduces data maturity models as one way to assess and improve data quality. Ours is narrower, since it covers only how a disagreement is found and fixed, and its levels are descriptive, not a standard. We have not measured how long a move between levels takes.

Each level can be checked by asking to see something: a file and a person, a calendar and a log with an owner's name on it, a count of records in, out and rejected for each hand-off, a register of interfaces with contracts and owners, and finally an alert that reached a person and a record of what that person did. The step that matters most is the first, because moving from a person to a calendar turns a favor into a duty.

EXHIBIT 8

From reconciled by hand to failures are loud

Each level can be checked by asking to see one thing: a file, a calendar, a count, a register, an alert.

What is the step-by-step response?**1. Choose the records that matter**

The executive owner, the finance or operations lead and the architect list the records whose disagreement hurts, for example the customer, the order, the product and the payment, and the figures the board reads. At the end you have a short list of records, the reports that depend on them and the decisions those reports feed.

2. Trace each record from entry to report

The architect, an engineer from each system and the person who prepares the report walk the route and list every entry point, copy, job and spreadsheet, including the manual hops. At the end you have a route map for each record with an owner beside each hop. A hop with no owner is a finding.

3. Write the definitions

The business owner of each term and the analysts gather the rules in use for the terms that matter, such as active, closed, revenue and customer, decide which are separate things needing separate names, and write each rule once. At the end you have a glossary: each term with a rule, an owner and the reports that use it.

4. Name the system of record

For each record, the business owner decides, on the architect's advice and using the questions in Exhibit 5, which system is accountable for its correct state. At the end you have a written declaration for each record, signed off by its owner, and a list of the copies that must become reads or be declared as copies.

5. Find the quiet failures

Engineers compare both ends of each hand-off directly instead of reading the job's status, looking for skipped, rejected, duplicated and late records, and test a retry and out-of-order arrival. At the end you have a list of gaps, each traced to its cause and ordered by damage.

6. Make failures loud

The interface owner and engineers add a count of records in, out and rejected to each hand-off, an alert to a named person, and handling for repeats and order, and remove any tolerance that skips records without an alert. At the end every hand-off reports its own state to a person answerable for it.

7. Start the reconciliation

The reconciliation owner, a person with authority to get a source fixed, schedules it, runs it by hand once to prove the keys and tolerances, then automates it. At the end you have a calendar, a log of results, a place where open differences can be seen, and a name.

8. Retire the workaround

The spreadsheet's owner and an engineer read it as a specification, move its rules to the source or into a check with tests, run both side by side once, and switch the spreadsheet off on an agreed date. At the end the knowledge no longer depends on one person.

9. Make interfaces products and keep reviewing

Each interface gets an owner, a contract, a version, monitoring and an end date, and a change to a definition updates the definition, the contract and the check in the same release. The leadership group reviews the log and open differences on a standing cadence. At the end you have a register, a rule for change and a review that continues after the project team has gone.

How can you check your own organization?

- For each record that matters, we can name the one system in which a correction must be made for it to hold.
- Each of those records has a named owner who approves changes to its definition.
- Each business term used in a board report has one written rule, and where two meanings are needed they have two names.
- Each integration reports how many records went in, arrived and were rejected, and a named person sees the report.
- A job that skips a record raises an alert and does not simply finish.
- Our receiving systems cope with the same message arriving twice and with messages arriving out of order.
- Two systems that hold the same fact are compared record by record on a calendar, and someone owns the result.

- A difference found in reconciliation is fixed at the source, not in the report.
- Each interface has an owner, a written contract and a way of telling its users when it changes.
- A change to a definition updates the definition, the checks and the documentation in the same release.
- Before an AI system reads company data, we know which system is the record for each fact it reads.
- A leader can say, for each figure in the last board pack, which system it came from and by which rule.
- No spreadsheet holds a rule that no system holds, unless it has a retirement date.

What do readers ask when their systems disagree?

Why do our systems give different answers about the same customer?

The same fact is entered or changed in several places, and no system is named as the record. Each team needed its own copy to do its job, definitions drifted, and an integration skipped records without a signal. Each fix was sensible alone.

Should every team use the same meaning of "active customer" or "revenue"?

Not necessarily. Each meaning can be correct by its own rule. Give each its own name, a written rule and an owner, and make every report say which meaning it uses. The trouble starts when a report adds different meanings together as if they matched.

What should we do about the spreadsheet someone uses to reconcile our figures?

Treat it as a specification, not as something to ban. Its tabs and columns are the rules the estate lacks. Move them to the source or into an automated check with an owner, run both side by side for a time, and switch the spreadsheet off on an agreed date.

Can we build AI or reporting on data that disagrees?

That depends on the purpose. A model does not settle a disagreement between systems. It reads either or both and answers with the same confidence. An estate may be good enough for a monthly summary and not for an automated decision, so name the record for each fact first.

Sources and notes

Each document was opened and read for the passages cited on 20 October 2026. Dates and versions are as published, and vendor documentation describes the service on the day it was read.

1. [The Government Data Quality Framework, Government Data Quality Hub, 3 December 2020. Read: the foreword, principles, data lifecycle and the six dimensions as defined by DAMA UK.](#)
2. [The Government Data Quality Framework: guidance on action plans and root cause analysis. Read: the advice to fix at source and to cost not fixing.](#)
3. [European Statistics Code of Practice, European Statistical System Committee, 16 November 2017. Read: indicator 12.1 and Principle 14, indicators 14.1 to 14.4.](#)
4. [New European Interoperability Framework, European Commission, annex to COM\(2017\) 134, 23 March 2017. Read: section 4.3.3 on base registries and recommendations 31, 37, 38 and 40.](#)

5. [Artificial Intelligence Risk Management Framework, AI RMF 1.0, NIST AI 100-1, January 2023. Read: Appendix B, how AI risks differ from traditional software risks.](#)
6. [Right to rectification, UK GDPR guidance, Information Commissioner's Office. Read: the account of Article 16 and the checklist item on informing recipients. The page says it is under review following the Data \(Use and Access\) Act.](#)
7. [Stripe documentation on receiving events in a webhook endpoint. Read: retries, event ordering, duplicates and returning success before complex logic.](#)
8. [Amazon Web Services documentation on Amazon SQS standard queues. Read: at-least-once delivery.](#)
9. [Microsoft Learn, fault tolerance of copy activity \(page dated 26 September 2024\). Read: continuing past incompatible data and optional logging.](#)
10. [Google Cloud, loading CSV data from Cloud Storage \(last updated 19 October 2026\). Read: the bad-records setting and its default.](#)
11. Nothing from a client engagement is cited. The delivery experience described in the findings is pattern and mechanism only.

USE THIS

[Assess the risks in your architecture](#)

[Use the architecture review checklist](#)

Choose one record that matters, such as the customer, and trace it from where it starts to where it is reported. Every hand-off you find is a place to add a check.

This document was made from the article at <https://xonik.com/insights/reports/when-systems-and-data-stop-agreeing/>. The online article is the current version.