

THE



WAY TO COMMERCIAL PHARMA DATA

Launch faster, spend less, and trust the numbers.

Christopher Bergh, Gil Benghiat, and Eric Estabrooks



DataKitchen

The DataOps Way to Commercial Pharma Data

Launch faster, spend less, and trust the numbers.

Christopher Bergh, Gil Benghiat, and Eric Estabrooks

Copyright DataKitchen, Inc. 2026

Table of Contents

Why commercial pharma data is different	4
How DataOps is Transforming Commercial Pharma Analytics	5
IQVIA, specialty pharmacy, and your own files: the commercial data most likely to be wrong	11
Why Testing Commercial Pharma Data Is Harder Than Anyone Tells You	18
The Illusion of Clean Data: The Pharma Identifiers Generic Tools Never Check	25
The launch	34
The Launch Data Timeline and the 24 Reports	35
Celgene: Meeting the Product Launch Challenge with DataOps	41
Drug Launch Case Study: Amazing Efficiency Using DataOps	44
Hoarding cash before the FDA gate opens? Your data and analytics bill is the easiest line to cut.	54
Who should run your commercial data	63
The \$100 Billion Secret: Why Leading Pharma Companies Bring In a Dedicated Commercial Data Team	64
The Age Of Outsourcing Your Commercial Pharma Data Team To India Is Over	72
You Built the Thing. Now You're Stuck Running It.	80
We Have a Cadillac, We Need a Corolla: Cutting a Biotech's Data Bill to Fund the Next Launch	87
Cut the Data Run-Rate on the Product You Already Sell	93
Why I Chose DataKitchen for DataOps	98
The process hub and the platform	100
Centralize Your Data Processes With a DataOps Process Hub	101
Large Pharma Achieves 5X Productivity Gain With DataOps Process Hub	108
Implementing a Pharma Data Mesh using DataOps	112
Webinar Summary: Data Mesh and Data Products	118
AI and technology	125
The Equation For AI Success: $DT + DX + CTX = 10x$	126
We've Been Using FITT Data Architecture For Many Years, And Honestly, We Can Never Go Back	141
DataOps + FITT + Data Testing = 10x Data Engineering Productivity with AI	152
Build a Data Mesh Factory with DataOps	172
Running it day to day	187
The Five Organizational Sins of Data Teams	188
How Do I Get My Data Team More Efficient?	196
Webinar Summary: Driving Data Analytic Team Excellence Through Agility, Efficiency, and Aphorisms	204

Introduction

Commercial pharma data is not like other data. The feeds arrive from vendors you do not control, on schedules that slip, with identifiers that look valid and are not. A prescriber gets duplicated and the incentive comp run pays the wrong rep. A specialty pharmacy file lands half empty during the weeks that matter most. And all of it lands on a clock, because the first six to twelve months of a launch set the trajectory a brand rides for the rest of its life. You get one chance at the field's trust.

The people asking for those numbers are demanding, and they should be. A brand lead who catches a bad number once stops trusting the dashboard for good. Commercial teams work with some of the most complicated datasets in any industry: prescriptions, claims, specialty pharmacy dispenses, payer hierarchies, field activity, and a dozen syndicated feeds that disagree about what a date means. They know their market and their customers extremely well. What they are not, and should not have to be, is data engineers. They should never have to know why the IQVIA refresh disagrees with the claims file.

The three of us have worked with commercial pharma data for more than two decades, and DataKitchen has spent more than ten years inside that problem. Seven of our engineers supported about \$10 billion in US sales at Celgene, with ten to 12 analysts covering hundreds of datasets and no missed SLAs. One and a half engineers built and ran the launch data platform behind Cobenfy at Karuna, across 50 integrated datasets. Three of our customers have been acquired for nearly \$100 billion combined. We did not get there with more people. We got there by treating the commercial data operation as a factory: one that tests every table on every run, deploys through automation rather than tickets, and catches a bad vendor file before the field ever sees it. That way of working is DataOps. It is the argument of this book.

The chapters here were written over ten years and gathered in one place for the first time. They cover why commercial pharma data fails in the specific ways it does, what a launch data platform has to deliver and when, who should run your commercial data and what it should cost, the process hub and platform underneath, and what AI changes about all of it. Some are case studies with the customer's own numbers. Some are arguments we expect you to disagree with. People who have run the standup, chased the vendor about the late file, and explained the restated number to the brand lead wrote every one of them.

Our tools are open source, our engagements are flat rate, and the code lives in your repository from the first sprint, so the platform is yours to keep or hand to whoever you name. We wrote this book for the same reason we work that way: the knowledge of how to run commercial pharma data well should not be locked inside a consulting contract. We hope you find ideas here that match what you have seen, and a few you can put to work before your next launch.

Christopher Bergh, Gil Benghiat, and Eric Estabrooks

This book was updated on September 24, 2026.

SECTION 1

Why commercial pharma data is different

1. How DataOps is Transforming Commercial Pharma Analytics
2. IQVIA, specialty pharmacy, and your own files: the commercial data most likely to be wrong
3. Why Testing Commercial Pharma Data Is Harder Than Anyone Tells You
4. The Illusion of Clean Data: The Pharma Identifiers Generic Tools Never Check

How DataOps is Transforming Commercial Pharma Analytics

Read the original article at: <https://datakitchen.io/blog/how-dataops-is-transforming-commercial-pharma-analytics/>



DataOps has changed how pharmaceutical analytics teams run. The teams that get it right ship better answers, faster, with fewer mistakes. That advantage shows up most clearly in product launches, where speed and accuracy translate directly to revenue. DataOps is about eliminating errors, cutting cycle time, building trust, and gaining agility.

After a decade of drug development, the launch is the most important phase in a product's life. The first 6 to 12 months sets the lifetime revenue trajectory. A solid early ramp puts a new medicine on track to hit its sales targets. A weak launch costs you revenue you'll never recover. Figure 1 shows how the curves diverge.

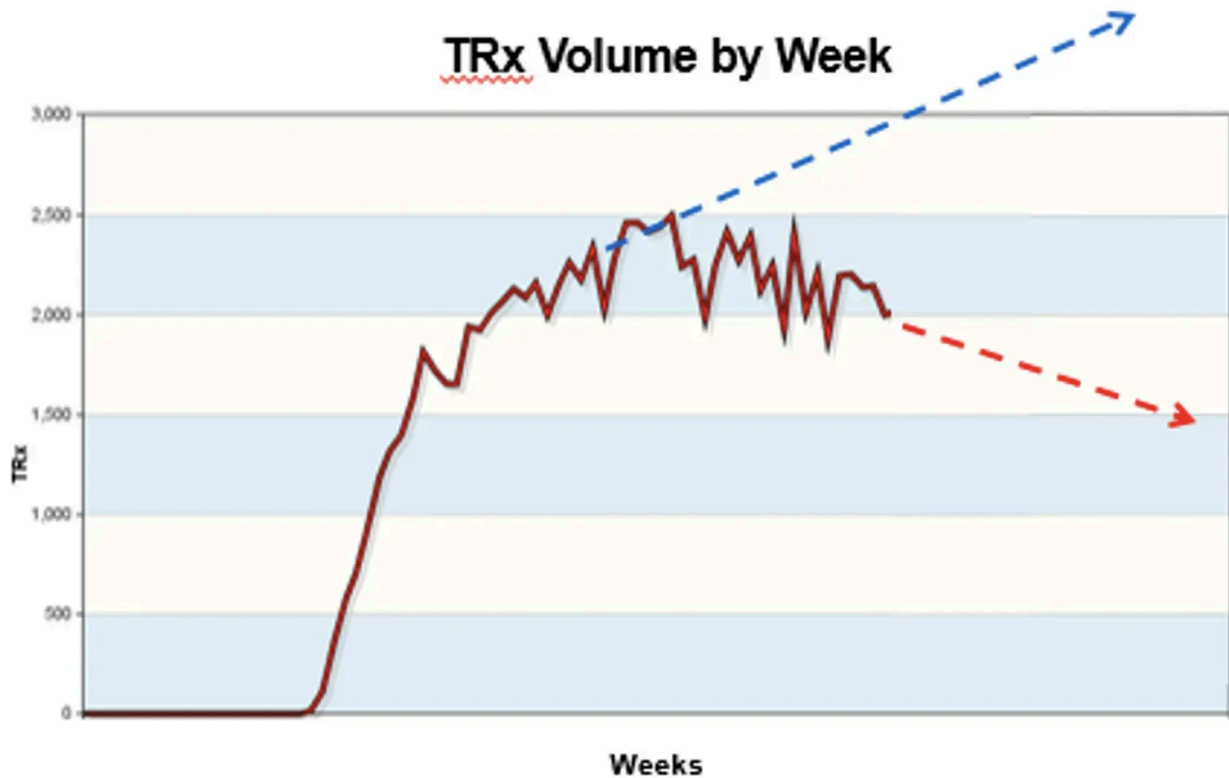


Figure 1: The first months in the product lifecycle determine its lifetime revenue.

During the launch, sales and marketing run on data. Campaigns are multi-level and analytics-driven. The work is so critical to product success that the data team often reports directly into sales and marketing.

Picture a data team of one or two dozen people serving hundreds of sales and marketing users. Each user has a territory or a campaign. They want new datasets, new dashboards, new segmentations, new cached extracts. The requests don't stop. The clock doesn't stop. You ship answers fast and you ship them right. The **24 recurring reports a launch analytics team owns** is the concrete version of that queue.

The insights flow hourly, daily, weekly, and monthly. Figure 2 shows how the data team ingests data from hundreds of internal and third-party sources.

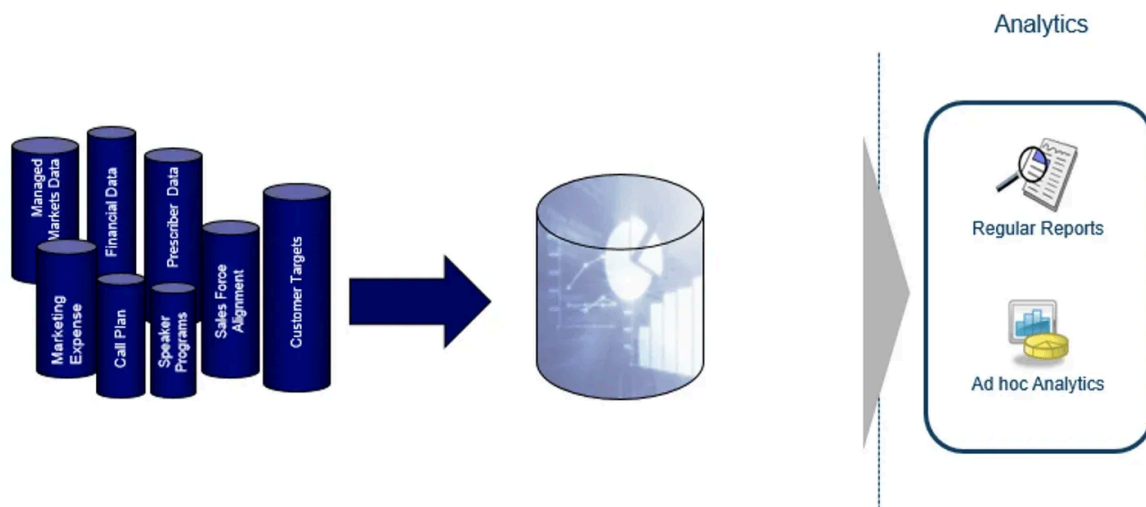


Figure 2: During the product launch, data comes from various sources and feeds into regular and ad hoc reports and analytics.

The data is messy. There are multiple lists of physicians and they don't agree. Roughly a million physicians practice in the US, but only 40,000 are targets for a given drug. Standardizing that list matters because sales compensation rides on it.

Part of the job is judging whether a source is fit for purpose. Figure 3 shows the spread: forecasts, stocking, sales, physician, claims, payer promotion, and finance.

#	Report / Activity	Description	Frequency	Data	Who	Key Metrics
2	Forecast Tracking Report	Report that displays key national metrics about brand performance against forecast	Weekly	NPA, Forecast Metrics	Sr. Mgmt, Brand Teams	TRx, NRx, Units
	Forecast Creating	Activity to develop forecast, which could be based on analogs, historical trends, population facts, projected events, or other methodology	Quarterly Update	NPA, Other Data Sets	Sr. Mgmt, Brand Teams, Finance	TRx, NRx, Units
3	Launch Tracking Report	Report at the Executive Level that summarizes key points from other, more detailed reports	Weekly	Multiple Data Sets	Sr. Mgmt, Brand Team, Rest of Home Office	TRx, NRx, Cohorts, Geographic Differences, Deciles, Specialties, Targets, Activity Actual vs. Plan, Payer Performance, Top Docs
4	Stocking Report	Report that displays which pharmacies carry the brand	Weekly	Symphony Stocking Data	Brand Team	Number of Pharmacies Stocked
5	Brand Team Report	Report that covers critical sub-national topics in depth	Monthly	Symphony Sub national, AMA Letter shop, Alignment, Targeting, Promotional, Field Activity, and Other	Brand Team	TRx, NRx, Cohorts, Geographic Differences, Deciles, Specialties, Targets, Activity Actual vs. Plan, Payer Performance, Top Docs, Promotional Execution, Promotional Results, Others
6	Field Activity Report	Report that displays sales representative activity compared to planned activity	Weekly	AMA Letter shop, Alignment, Targeting, Field Activity	Field	Activity (calls, samples)
7	Field Performance Report	Report that displays brand performance by geographic alignment	Weekly	Symphony Sub national, AMA Letter shop, Alignment, Targeting	Field	TRx, NRx, Activity (calls, samples)
8	Pre-call Planning	Physician level reports that integrate multiple data sets	Weekly	Multiple Data Sets	Field	Multiple Metrics by Physician
9	IC Report	Report that ranks each sales representative based on performance and business logic	Weekly	IC Grade Symphony Sub national Data	Field	IC Metrics, Rank
10	Supporting Plan of Action (POA) / National Sales Meetings	Activity to develop series of presentations, interactive quizzes, posters, and facilitated sessions	At Meetings	Multiple Data Sets	Field	Field-Based Activity and Performance
11	Payer/Plan Report	Report that displays performance of Major Model Groups, Minor Model Groups, and Individual Payers/Plans	Monthly	Plan track, Formulary, Co-pay, with Bridge File	Brand Team	TRx, NRx, Units
12	Claims Reports	Reports on numbers of claims per product over time	Weekly	Symphony Claims Data	Sr. Mgmt, Brand Team, Field	Multiple Metrics
13	Brand Planning	Activity to support brand team over few months of annual brand planning	Few Months, Annually	Multiple Data Sets, Drivers Analysis	Sr. Mgmt, Brand Team	Multiple Metrics
14	Responding to Analytic Questions	Activity to respond quickly and efficiently to questions about the launch and other sub-national market dynamics	On-Tap	Multiple Data Types	Extended Sales & Brand Team	Multiple Metrics
15	Promotional Mix / ROI Report	Report that displays lift analysis and ROI calculation for major promotional efforts	Monthly	Multiple Data Sets	Brand Team	TRx, NRx, Units, Financial Return
16	DTC Report	Report that displays DTC geographic coverage and displays lift analysis	Monthly	DTC Coverage, Symphony Sub national Data, Alignment	Brand Team	TRx, NRx, Units
17	Non-personal Promotion (NPP) Report	Report that displays lift analysis and ROI calculation for major non-personal promotional efforts	Weekly	Promotional Data, Symphony Sub national Data, Field Activity Data	Brand Team	clicks, views, reads, etc.
18	Source Of Business Patient Level Reports	Report(s) that provide insights about a patient adherence, refills and sales	Monthly	Source of Business / Longitudinal Patient Data	Sr. Mgmt, Brand Team	Multiple Metrics
19	Specialty Pharmacy Report	Report(s) that provide insights about a specialty pharmacy stocking / sales	Periodically	Specialty Pharmacy Data	Sr. Mgmt, Brand Team	Multiple Metrics
20	Finance Report (s)	Report(s) that display insights about accrued financial metrics	Monthly	Finance Data	Sr. Mgmt, Brand Team, Finance	Multiple Financial Metrics
21	Demand based P&L Report	Report that uses TRx, SKU, pricing, contracting, and other data to build a "demand" based P&L	Monthly	Multiple Data Sets, including Contracting and Pricing Data	Sr. Mgmt, Brand Team, Finance	Multiple Financial Metrics
22	REMS (Risk Evaluation and Mitigation Strategy) Report(s)	Reports on use within certified components of delivery chain	Daily/Weekly/Monthly	REMS Specific Data Sets	Brand Team	Multiple Metrics for REMS
23	Developing Target List	Activity to identify most promising existing or potential prescribers	Periodically	AMA Letter Shop, Alignment, Symphony Sub national Data, Other Data Sets	Sales Ops	Scores by Physician, Zip to Terr
24	Supporting Periodic Sales force Logistics	Activities to enable sales force to function effectively and efficiently	Periodically	Multiple Data Sets	Sales Ops	Multiple Metrics

Figure 3: The vast and varied types of analytics required during the launch phase.

DataOps Success Story

The Otezla launch is a billion-dollar example of DataOps handling that complexity. Using DataKitchen DataOps Automation, the team ran an agile shop with seven data engineers and 10 to 12 analysts. They mastered hundreds of datasets serving thousands of users with very few errors or missed SLAs. They built tens of thousands of automated tests checking data and analytics. They shipped hundreds of schema and dataset changes a week without breaking anything. Compare that to companies running hundreds of data professionals who still struggle. That's the power of DataOps automation.

Figure 4 shows the architecture from a real biologic launch. Over seventy data sources, daily and weekly cadences. Biologic data is dirty, so the platform profiles it and runs rule-based quality checks first. Did the data arrive on time? Is the volume right? When tests pass, orchestration admits the data to the catalog.

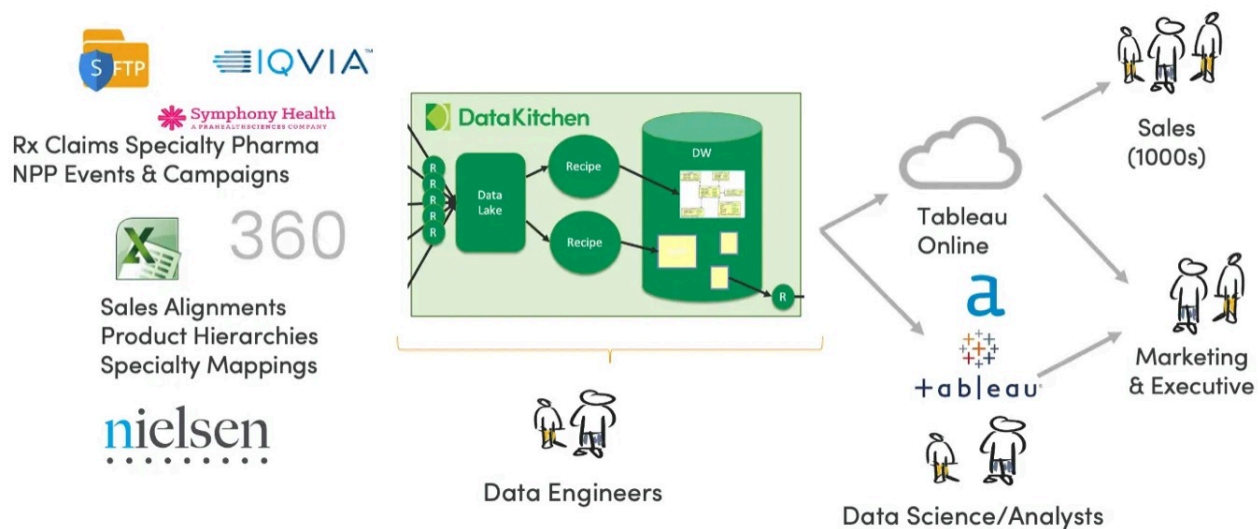


Figure 4: DataOps architecture based on DataKitchen DataOps Automation

Reporting schemas update several times a day. Purpose-built warehouses, marts, and star views get tailored to analyst needs. When builds finish, notifications trigger refreshes of dashboards, Tableau workbooks, and whatever reports the business unit requires.

The agility shows up two ways. Production analytics refresh several times a day from dozens of asynchronous sources. And the data engineers and scientists change any part of the pipeline at any time. New datasets, new schemas, new views. The Otezla team made hundreds of schema and dataset changes a week. Error rates stayed negligible because tests run at every stage. Observability catches problems before they hit the people consuming the analytics.

The platform replaces the army of people running manual tests, checklists, and procedures. Your team redeploys to work that creates real analytics. You ship more insight, faster. The table below shows features and benefits of DataKitchen DataOps Automation.

FEATURE	BENEFIT
Process and workflow automation that operates on data designed for analytics	Significantly faster time-to-insight for ad hoc requests
Automating (and reusing) process with a button push	Significantly fewer dollars spent on analytics consultants and staff
Enablement for analytics, not just IT infrastructure	Extend IT investment, not replace

When the processes around the data are built for rapid refresh and rapid response, [time-to-insight](#) drops and ad hoc requests cost less. DataOps automation replaces the non-value-add work and the outside consultant dollars with a framework that runs efficiently and at quality. You build a factory that produces insights.

The DataOps Platform doesn't replace your data lake or data hub. It's the automation layer that spans the toolchain you already have. Informatica, Snowflake, Databricks. DataKitchen works alongside them and stitches them into the end-to-end data lifecycle.

The process-centric view says this: *the processes that act on data are as important as the data itself*. Neglect those processes and you get delays, errors, and unplanned downtime. Build for them and you get faster, more flexible analytics with almost no downtime.

DataOps Is the “New Normal” in Pharma

Otezla used DataOps automation with DataKitchen to make changes fast and ship high-quality production deliverables. They built analytics consistently across dozens of product teams. They deployed pipelines at scale, across a large group of people, spanning many tools.

DataOps is about quality assurance, automation, reusability, and repeatability. The goal is analytics agility, and analytics agility is what gets you business agility. When the analytics team works fast and accurately, decision-makers get what they need. That's it.

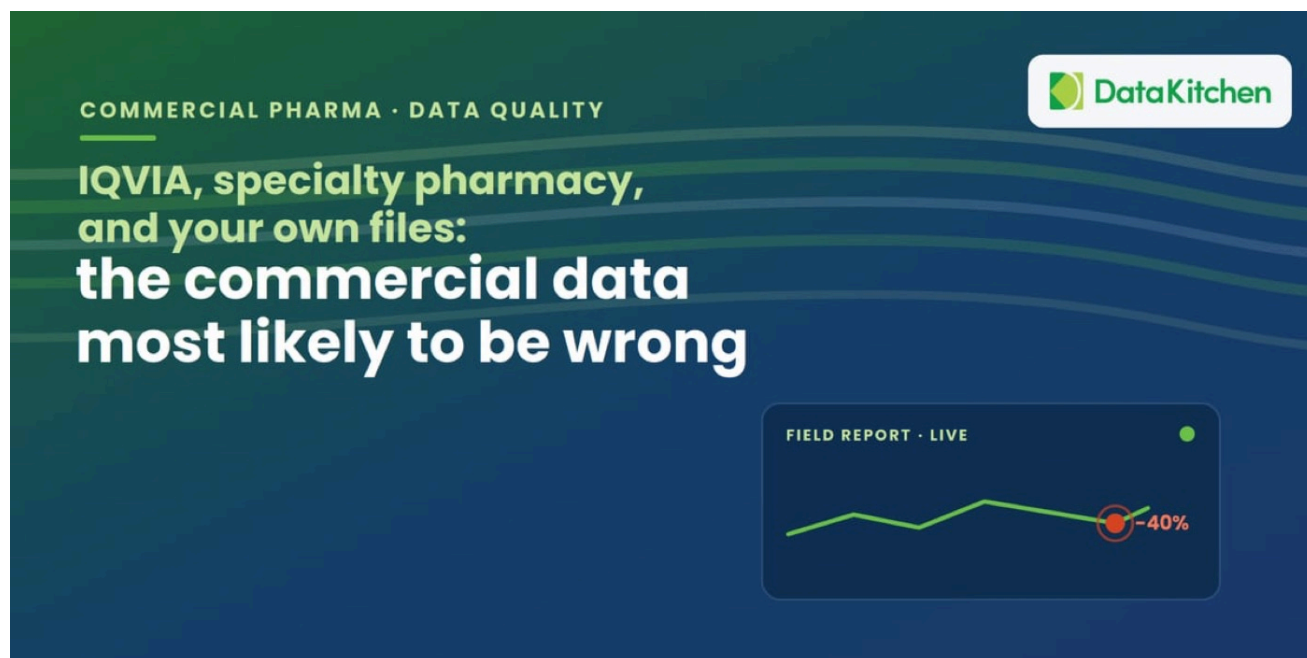
Want more launch stories? Read [How Three Small Pharma Companies Used DataKitchen to Achieve Commercial Launch Success and Skyrocket to \\$100 Billion in Acquisition Value](#).

Curious how DataOps can improve pharma R&D operations? Read [Accelerating Drug Discovery and Development with DataOps](#).

This post was updated on August 27, 2025.

IQVIA, specialty pharmacy, and your own files: the commercial data most likely to be wrong

Read the original article at: <https://datakitchen.io/blog/commercial-pharma-data-most-likely-to-be-wrong/>

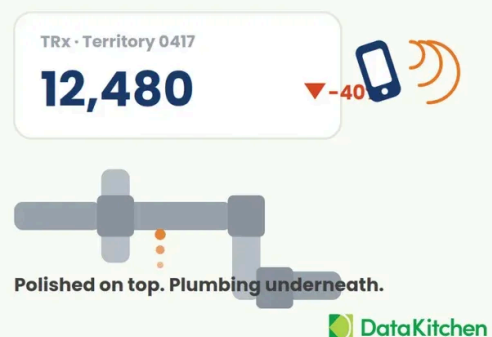


Most of the data behind your launch came from someone else. IQVIA sent Xponent. Symphony sent claims. The specialty pharmacies sent dispense data on their own schedules, in their own formats, with their own idea of what a “week” is. You load all of it. Then you build the field report on top and hope the numbers hold.

They don’t always hold. And the way you find out is a rep calling to ask why their territory dropped 40 percent overnight. Testing commercial pharma data is **harder than anyone tells you**, and it starts with the feeds themselves.

WHO GETS THE CALL

You own the number, not the plumbing



You own the number, not the plumbing

When the number is wrong, the brand team does not call your data engineer. They call you. And while that fire burns, your analysts spend the week reconciling files by hand rather than identifying the next access opportunity. The problems below look technical. Every one of them is yours. It helps to know what they look like.

COMMERCIAL PHARMA DATA

The landmines

NPI	PRESCRIBER	TRx	SRC
⚠️ 1043827	J. Okafor	842	IQVIA
⚠️ 1043827	J. Okafor	842	SYM
2290144	R. Patel	1,203	IQVIA
7781902	M. Alvarez	⚠️ 98,400	IQVIA
5560318	L. Chen	671	IQVIA

DataKitchen

The landmines

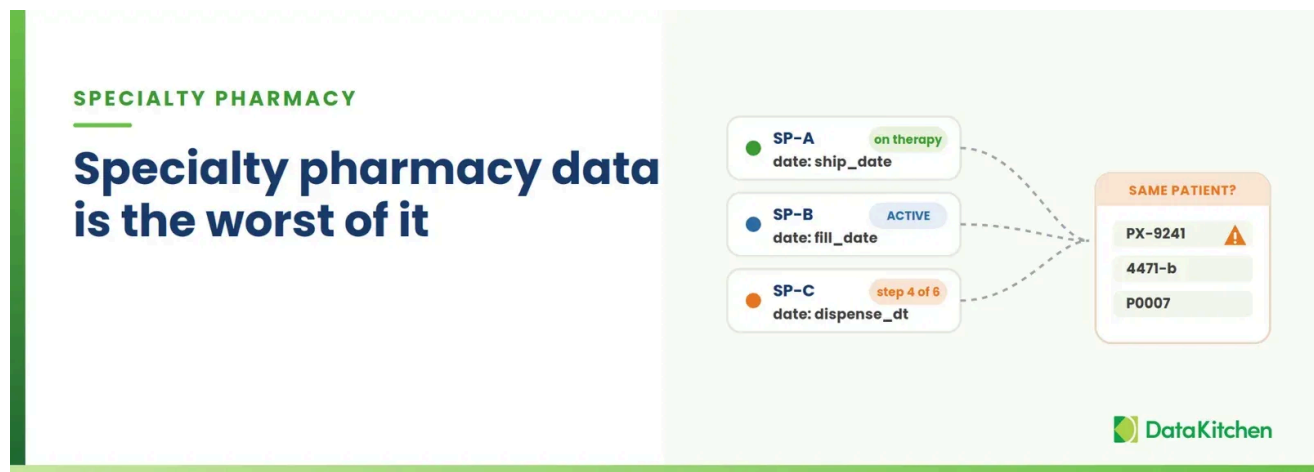
Two physician files, one truth, and they disagree. You get prescriber data from multiple sources. IQVIA says Dr. Smith wrote 30 scripts. Your MDM says Dr. Smith and Dr. Smith are two different people. Now your targeting list has a ghost and your alignment double-counts. Nobody notices until the incentive comp run, and then everybody notices. A transposed NPI does the same damage from the other direction, and **it passes a ten-digit pattern check every time.**

The vendor restated last quarter without telling you. Third-party sales data is part actual, part projection. The vendor backfills the gaps with an algorithm. Next month they get better inputs and rerun it. Your actuals from last quarter quietly changed. The trend line you showed the VP last week is now different. Same query, different answer.

The decimal moved. A quantity column arrives as units one month and as packages the next. Or a price lands in cents instead of dollars. One product shows a 10x jump, and the dashboard renders it as a heroic sales week. It was a unit-of-measure change nobody flagged.

Half the file showed up. The specialty pharmacy feed is supposed to have 40,000 rows. Today it has 26,000. The load succeeded. No error. The pipeline is green. Your dispense numbers are just wrong, and everything downstream inherits the hole.

The vendor renamed a column. Last week it was RX_CNT. This week it's RxCount. Your transform maps the old name, finds nothing, and fills the field with nulls. The pipeline doesn't fail. It produces a clean, well-formatted, completely empty metric.



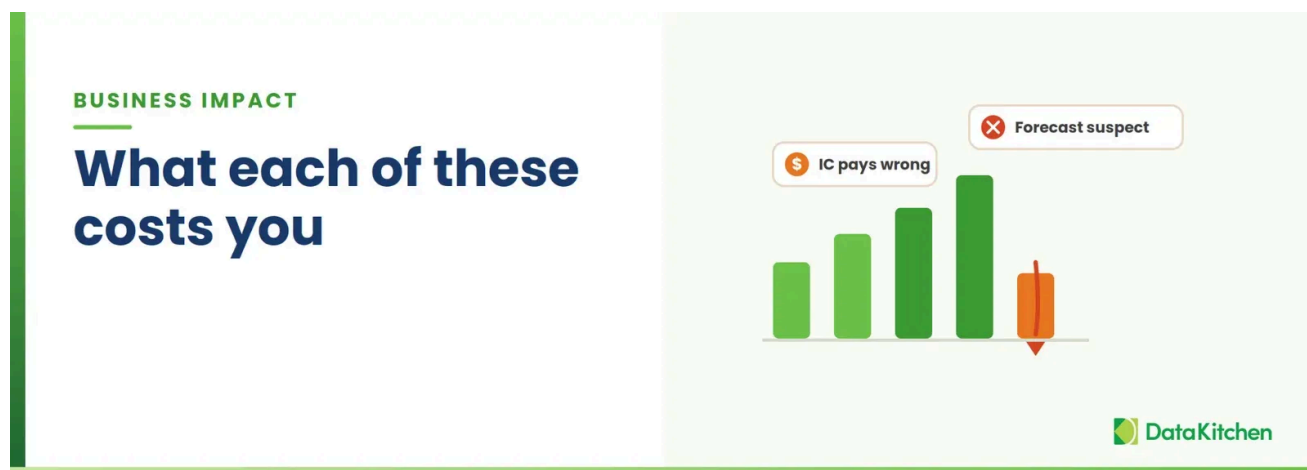
Specialty pharmacy data is the worst of it

If the vendor feeds are messy, specialty pharmacy data is the mess with another mess stacked on top. A retail brand pulls prescription data from one or two big aggregators. A specialty brand pulls it from the network of specialty pharmacies that actually dispense the drug, and that network can be a dozen pharmacies or more. Each one sends its own file, on its own schedule, in its own format.

None of them agree on what the words mean. One pharmacy's ship date is another's fill date is another's dispense date. One counts a referral the day it arrives, the next counts it the day benefits clear, the third counts it when the first box goes out the door. Patient status is worse. One calls it on therapy, another calls it active, a third has six sub-statuses you have never seen and no way to map them to anyone else's. You are supposed to stitch all of it into one patient journey, and the pieces were cut by different hands.

Then there is the patient key, or the absence of one. Specialty pharmacy data is often your only window into patient-level activity, because a limited-distribution drug has no retail claims to fall back on. But each pharmacy de-identifies patients its own way, so the same patient can show up three times under three tokens and you cannot tell. Your adherence rate, your time to fill, your discontinuation curve all sit on top of that.

Here is the part that makes it dangerous. Nobody owns this data. It lands on an analyst who is already doing three other jobs, who opens the files, does their best in a spreadsheet, and moves on. Specialty pharmacy integration is almost always somebody's afterthought, right up until it becomes the number the access team is betting the launch on.



What each of these costs you

None of these is a rounding error. A duplicated prescriber splits credit across two records, and your incentive comp run pays the wrong reps. A silent restatement moves a number you already briefed to leadership, and every forecast built on it is now suspect. A unit-of-measure jump sends the field chasing a spike that never happened. A half-loaded specialty feed hides real patients from the access team during the weeks that matter most. Every feed a commercial launch runs on fails in its own particular way, and we catalogue them in a [feed-by-feed inventory of commercial pharma data sources](#).

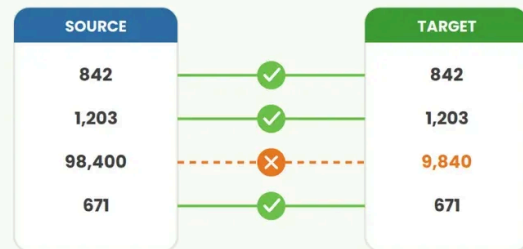
Launch is where this hurts worst. The first year sets the trajectory a brand rides for the rest of its life, and you get one chance at the field's trust. One wrong territory number and the reps stop believing the dashboard. After that, they run on gut, and the platform you paid for is furniture. Good [commercial pharma analytics](#) starts with the data the field believes in.

There is a second cost, quieter than the first. Reconciled data is defensible data. When a rep disputes their comp, you need to show the number held. When leadership challenges a forecast in the business review, you need to show the history did not move under you. And if your company is the acquisition target your investors are hoping for, the data behind your commercial story has to survive due diligence. A reconciliation trail on every feed is what turns "trust me" into "here is the check that passed."

It does not have to go this way. One commercial launch platform, behind a brand later acquired for \$14 billion, ran on roughly one and a half engineers and had field reporting live six months before launch day, because the checks caught the bad loads before anyone downstream saw them. The [launch case study](#) has the numbers. The difference was not heroics. It was reconciliation on every feed.

TWO CHECKS, NOT ONE

Test instead of trust



DataKitchen

Test instead of trust

You have to check two things, not one: that the file arrived intact, and that last month's history did not change underneath you. Most teams check the first and skip the second. The rest of this section is how.

The first check is extract to load. Did the file arrive intact? This one is deterministic, and it should be exact. Count the rows in the vendor's control record and in your staging table; if they match, the load is correct; otherwise, the load is wrong. Sum the amount column on both sides to the penny, at fixed precision, so two engines don't disagree over floating point. Confirm the max load date is today, not three days ago. When the vendor gives you no control count, fall back to an expected range instead of a fixed number, because a specialty feed legitimately shrinks on a holiday week. A count of 26,000 against a normal 40,000 is not automatically wrong, but it is automatically worth a look.

These checks run in seconds and catch partial, stale, and truncated files before anyone opens a dashboard. When one fails, the load stops in staging and never reaches the mart. That is the point of a **two-stage architecture**: bad data needs somewhere to fail that the field never sees.

Row counts and sums tell you the totals line up. They don't tell you that the rows are the same. Two files can carry identical counts and identical sums and still be full of swapped values. To catch that, you fingerprint each row in the source, fingerprint it in the target, and compare them. If a value changed, the fingerprint changes, and you see exactly which rows drifted.

THE SILENT RESTATEMENT

The check most teams skip



Same quarter. Different number.



The check most teams skip

Extract-to-load reconciliation assumes last month's history remains last month's history. In commercial pharma, it often does not. Third-party sales data gets rerun when the vendor receives better inputs, and your Q2 actuals change inside a file that loads perfectly. Source and target agree. Every deterministic check passes. And the number you briefed to leadership is now different.

The only way to catch that is to remember what the vendor told you before. Keep a snapshot of each period as you received it. When the next file lands, diff the overlapping periods against the snapshot. Expected corrections get reviewed and accepted. Silent rewrites get flagged before they reshape a trend line under you. This is the check most teams skip, and it is the one that protects your credibility with leadership. A commercial insights leader at Celgene reduced errors to about one per quarter and held it there for years. That is what disciplined reconciliation buys you.

YOUR CEO WANTS AI

The foundation your AI actually needs

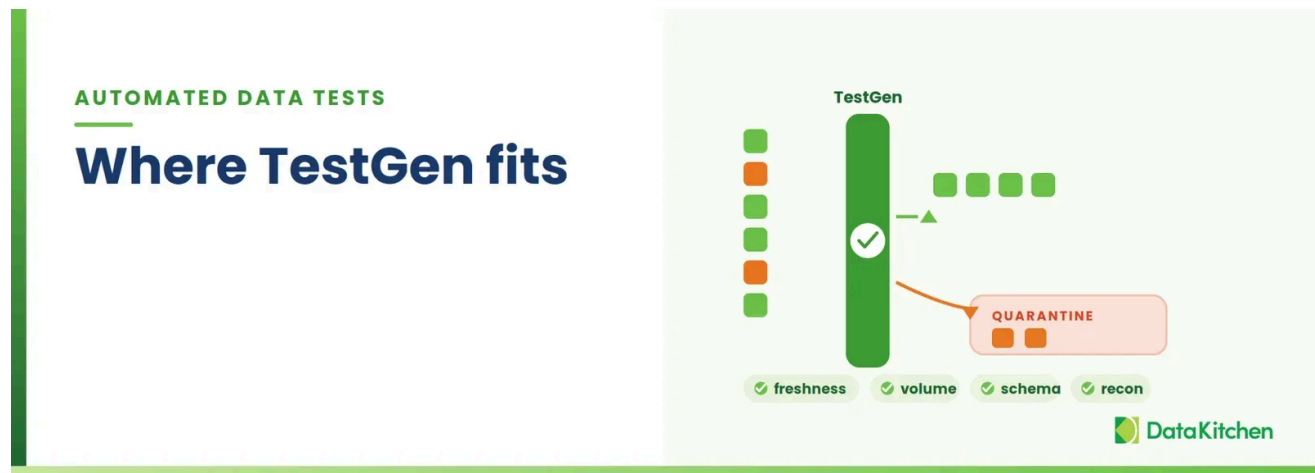


The foundation your AI actually needs

Your CEO wants AI, and you are the one who has to deliver it. Here is the part nobody says out loud. Your Snowflake Cortex Analyst or Databricks Genie cannot answer a real commercial question until the underlying data is right. Ask it for share by segment on top of an unreconciled feed, and it will answer with

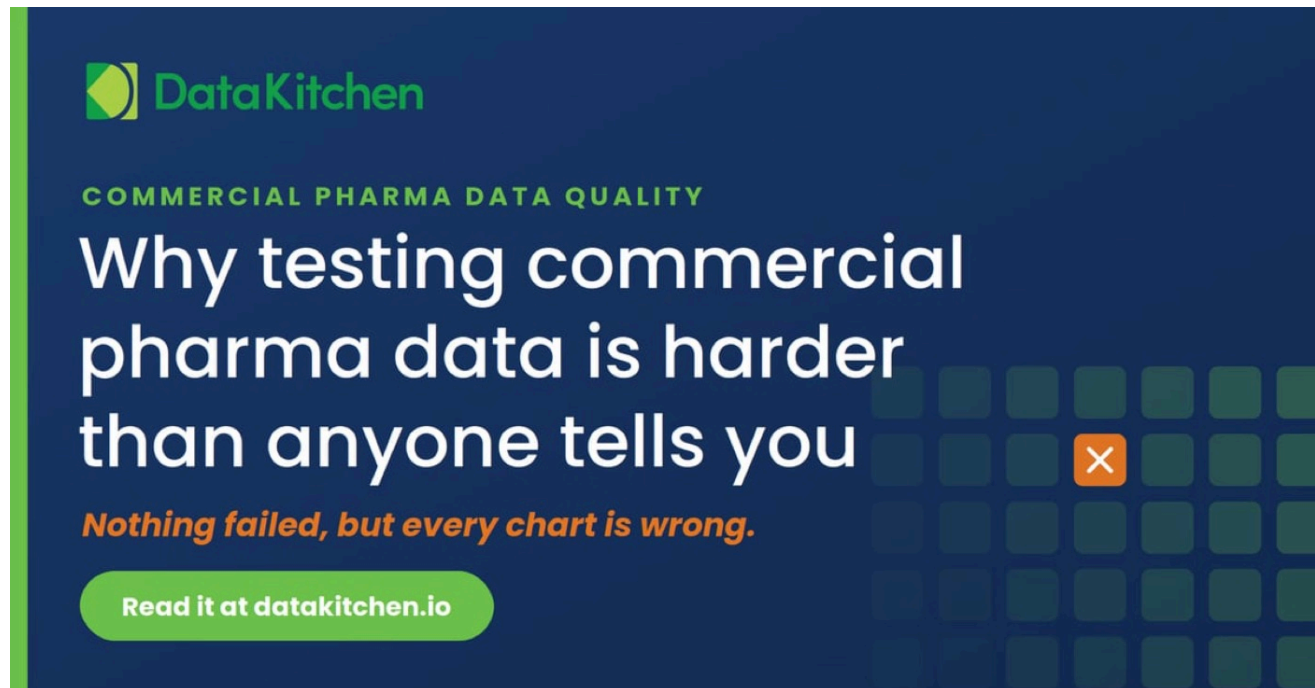
total confidence and be wrong, which is worse than not answering at all.

Reconciliation is not separate from your **AI mandate**. It is the first step of it. Trust in the data is what turns an AI demo into an AI answer, and it is the **Data Trust half of AI success**.



Why Testing Commercial Pharma Data Is Harder Than Anyone Tells You

Read the original article at: <https://datakitchen.io/blog/why-testing-commercial-pharma-data-is-harder-than-anyone-tells-you/>



You run data for a **commercial pharma** brand team. Your platform pulls in CRM activity, specialty pharmacy dispense feeds, claims, patient services records, REMS data, 3PL reports, and a stack of **syndicated datasets you pay a lot of money for**. Every one of them lands on its own schedule, in its own shape, keyed on its own identifiers. And every Monday, someone asks why the market share number moved.

Generic Data Quality vs. Commercial Pharma Reality

Generic Data Quality	Commercial Pharma Reality
Column validation (“Check for nulls”).	Relational integrity across fragmented sources.
Regex formatting (“Looks like a number”).	Spec validation & active list confirmation (Luhn check, NDC history).
Loud SQL load failures.	Silent downstream metric shifts surfaced in QBRs.

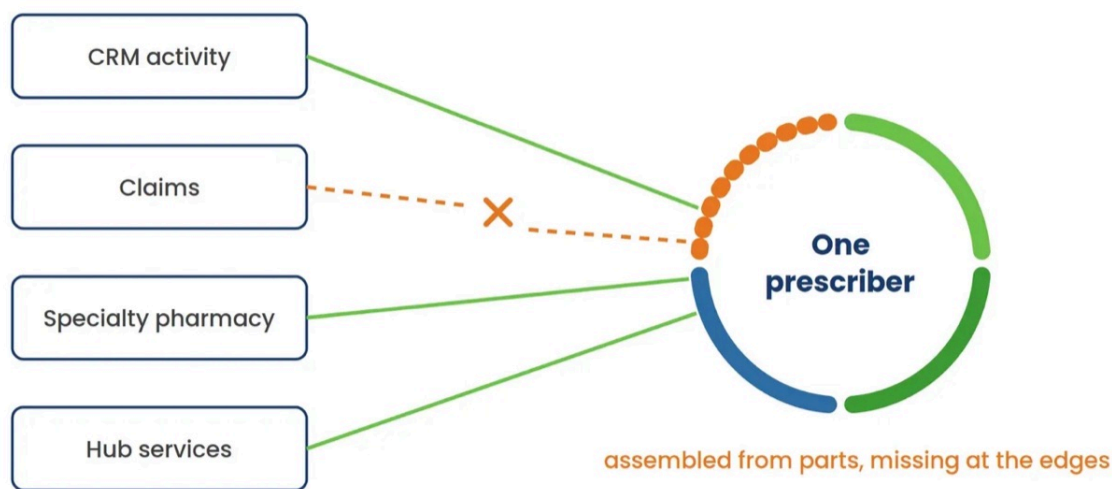
Generic data quality advice doesn’t help you much here. “Check for nulls” is not the problem. The problem is that **commercial pharma data** fails in specific, recognizable ways, and most testing approaches were built for a different data shape. Here are the challenges that actually bite, and at the end, a few things DataOps TestGen does about them. If you want the whole estate in one place, we keep a **feed-by-feed inventory of commercial pharma data sources** with the characteristic failure of each one.

Challenge 1: No single source is complete, so failures are relational

The picture of a prescriber, an account, or a patient gets assembled from several systems. Each one holds part of the truth. Each one disagrees with the others at the edges.

CHALLENGE 1

No single source is complete



Every value is valid. The relationship between them is broken.

That means the failures that hurt you are relational, not scalar. A territory ID in the roster that doesn't exist in the hierarchy. An account with sales activity that never made it into the account master. Two files from the same vendor run that disagree on their shared keys. A patient recorded in two active treatment states at once.

Column-level checks never see any of this. Every individual value is valid. The relationship between them is broken. If your testing stops at "is this field populated and formatted," your integration layer is running blind.

And absence matters as much as presence. A provider withdrawn under a deletion request must remain gone. Suppressed entities and blocked identifiers must not reappear in a refresh. One cleanup pass doesn't enforce that. Only a continuous check does.

Challenge 2: Syndicated data fails in predictable ways that generic checks can't see

Purchased data feeds your most important analytics and produces your most characteristic failures. Feed for feed, it is **the commercial data most likely to be wrong**, because you did not create it and cannot control its schedule or its format. The failure modes repeat.

A weekly file should arrive exactly once per week. A monthly file, once per month. Feeds go missing. Feeds get loaded twice. A delivery arrives at half its normal size because the vendor had a bad extract. None of this throws an error. The load succeeds. The numbers are wrong.

Then there's restatement. Many syndicated products revise the most recent months on every refresh, while older periods remain fixed. So a naive "nothing should change" rule fires constantly in normal behavior. Ignoring change entirely misses real breaks. You need checks that distinguish between an expected restatement window and a genuine problem in the stable tail.

Shape shifts are the sneakiest. A sudden change in the mix of payer types, specialties, or geographies usually means a mapping regression upstream, not a real market movement. By the time a brand analyst notices, the wrong number has been in three meetings.

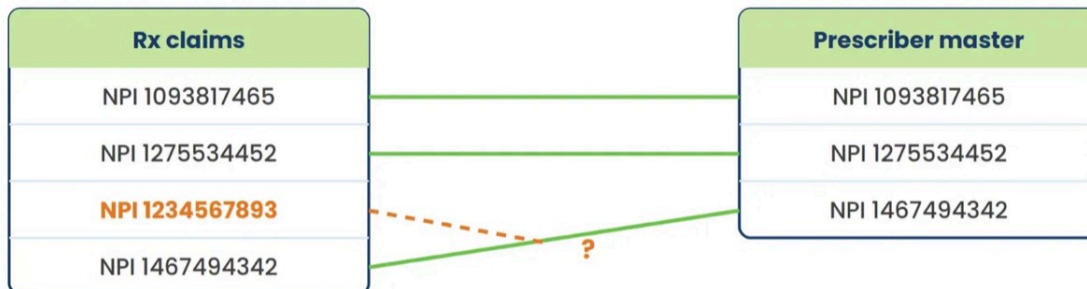
One practical trap on top of all this: vendors love wide formats, with time periods spread across columns. Wide files defeat most trend logic. Normalize them into long format at your refined layer before you try to monitor volume, freshness, or distribution. Test after the normalization, not before.

Challenge 3: Identifiers that fail silently

NPI, NDC, DEA, ICD, HCPCS, and CMS place-of-service codes are the join keys of the entire commercial pharma ecosystem. Here's the ugly part: a malformed identifier doesn't error. It silently fails to join, and the damage surfaces downstream as a quietly wrong number with no obvious cause.

CHALLENGE 3

Identifiers that fail silently



The pattern matches. The Luhn check digit does not.
No error thrown. The row silently drops from every joined metric.

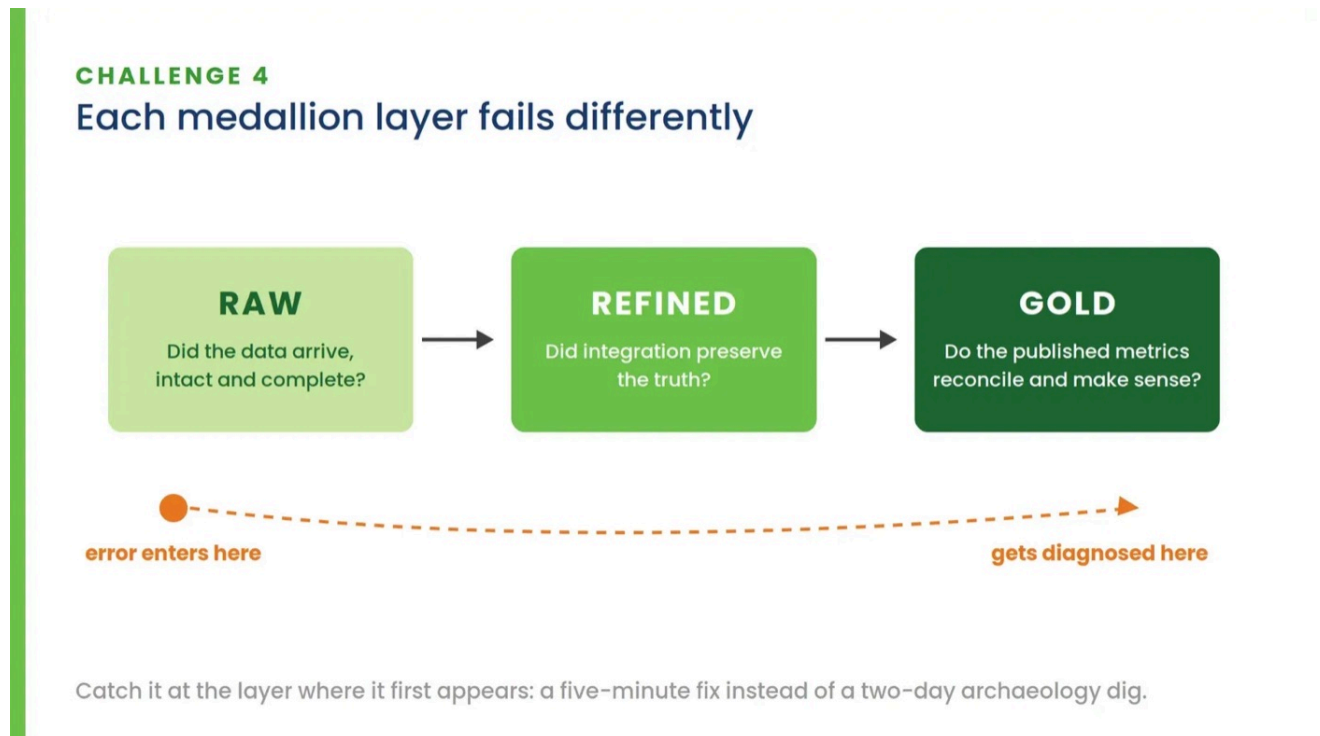
A regex confirms it looks like an NPI. It cannot confirm it is one.

A regex can't save you. NPI carries a Luhn check digit. DEA has its own checksum. A pattern match confirms the value looks like an NPI. It **cannot confirm the value is a valid NPI**. ICD-9 and ICD-10 look similar enough to confuse a generic format rule. And validity against the spec is only half the question. Whether an NDC is an active product, or a diagnosis code is in the current published set, requires checking against an authoritative external list that changes over time.

Most teams handle this with **custom SQL that one person wrote in 2022**, and nobody wants to touch.

Challenge 4: Each medallion layer fails differently

You've probably converged on some version of a **medallion architecture**: raw landing tables, refined and conformed layers, and gold datasets the business consumes. Testing only at the end means every failure surfaces at the layer where it's hardest to diagnose.



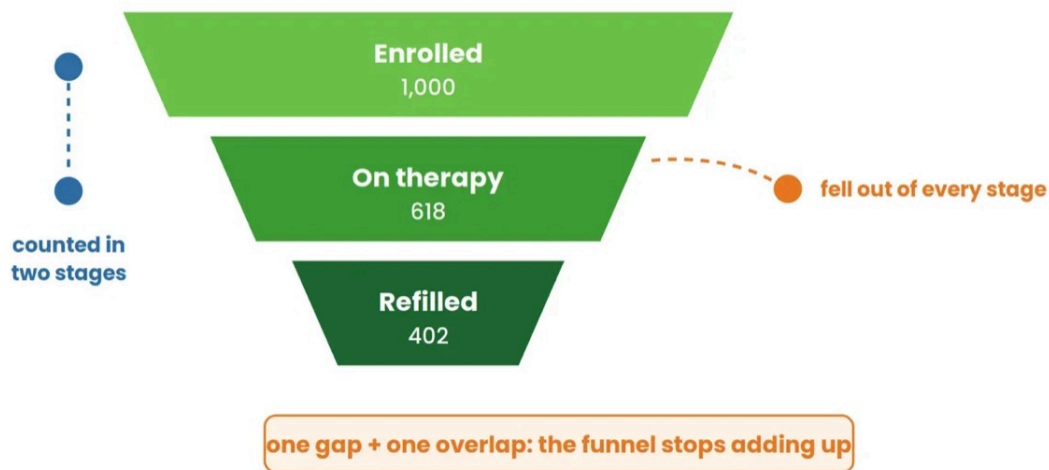
Each layer asks a different question. At raw: did the data arrive, intact and complete? In the refined layers: did integration and transformation preserve the truth? At gold: do the published metrics reconcile and make sense? A row count check at gold tells you something is wrong. It tells you nothing about which of forty upstream steps broke it. Catching the error at the layer where it first appears is the difference between a five-minute fix and a two-day archaeology dig.

Challenge 5: Gold datasets have internal logic, and that logic breaks

The gold layer is where quality problems cost the most because they're least visible. The columns are all populated. The formats are all valid. The logic is wrong.

CHALLENGE 5

Gold datasets have logic, and logic breaks



Every column is populated. Every format is valid. The logic is wrong.

Waterfall and funnel metrics require that every record be in exactly one stage. A gap in the classification logic and records fall out of the funnel. An overlap and you double count. Either way, the funnel stops adding up, and someone finds out in a QBR.

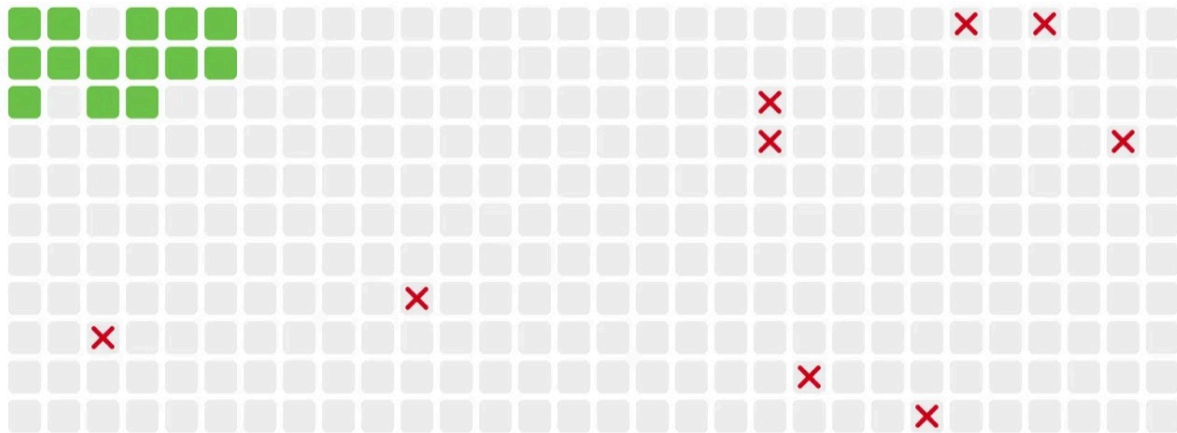
Patient journey and treatment views need milestone dates in a valid sequence. A ship date before an enrollment date isn't a format error. It's a logic error, and only a test that understands the sequence catches it. Same story with published totals: they have to tie back to source detail, per grouping, on every run. The day an integration step starts dropping or double-counting rows is the day you want to know, not the quarter.

Challenge 6: You can't hand-write your way to coverage

Here's the challenge underneath all the others. A single commercial data platform spans dozens of sources and thousands of columns, and the surface grows with every new feed, vendor, and brand. Hand-writing a test per column is not a realistic plan. So you write tests where you expect trouble: the handful of fields someone remembered to guard.

CHALLENGE 6

You can't hand-write your way to coverage



■ the columns someone remembered to guard ✕ where the errors actually show up

Thousands of columns. A handful of hand-written tests. Errors don't respect the plan.

Errors don't respect that plan. They show up in the long tail of columns nobody covered, and inside integration flows too intricate to script checks for by hand. Exactly the parts of the pipeline your manual test suite leaves dark.

The hand-written suite decays, too. Those hardcoded "expect at least 50,000 rows" thresholds go stale as the data grows. They either fire false alarms or get commented out. Both outcomes end the same way: you learn about the problem via a Slack message.

The Illusion of Clean Data: The Pharma Identifiers Generic Tools Never Check

Read the original article at: <https://datakitchen.io/blog/npi-ndc-icd-validation-tests/>

1234567890		167890123456		1234567890	
1234567890		167890123456		1234567890	
1234567890		167890123456		1234567890	
1234567890		1234567890		1234567890	
1234567890				1234567890	
123	THE ILLUSION OF CLEAN DATA				4567890
123	Why Generic Profiling Fails Healthcare Identifiers.				4567890
1234567890	1234567890	1234567890	12		
1234567890	1234567890	1234567890	12		

Picture the ticket. A rep writes in on Monday because their territory is short about 40 TRx and they want to know where it went. You trace it to one prescriber whose NPI in the **specialty pharmacy feed** does not match the NPI in Veeva. The value is not missing and it is not null: ten digits, all numeric, sitting there looking exactly like an NPI. Two of them are transposed, and the join has been dropping that prescriber since March. Your pattern test on that column has been green the entire time.

It was green because commercial pharma runs on identifiers almost nobody outside the industry has heard of. NPI for the prescriber. DEA for the one who can write a controlled substance. NDC for the product, at two different levels of specificity. ICD-9 and ICD-10 for the diagnosis, often in the same column. HCPCS J-codes for anything infused. CMS Place of Service for where it was given. NCT for the trial. These are the join keys the whole commercial picture is assembled on, and every one of them has a published format with rules that a general-purpose data quality tool knows nothing about.

So it counts characters instead. Ten digits in the NPI column, so the column passes. That is the hole, and it is wide enough to drive a quarter through.

The pack, in one paragraph

As of 27 August 2026, [DataOps TestGen](#) Enterprise ships a generation set called **Commercial Pharma**: twelve test types covering the identifiers above, validated the way their specifications define them rather than by counting characters. They generate off your profiling run alongside the standard set, and there is nothing to configure. The [release notes](#) list the whole set.

Test	What it checks	Severity
Valid NPI	10 digits plus Luhn check digit with the 80840 prefix	Fail
Valid DEA	Letter, letter or 9, seven digits, with the DEA check digit	Fail
Valid NDC	11-digit or hyphenated package form (5-4-2, 4-4-2, 5-3-2, 5-4-1)	Warning
Valid NDC Product	Hyphenated two-segment product form (4-4, 5-3, 5-4)	Warning
Valid NDC Any Form	Any recognized NDC form, including bare 10-digit	Warning
Valid ICD-10 Diagnosis	ICD-10-CM shape, dotted or undotted	Fail
Valid ICD-9 Diagnosis	ICD-9-CM numeric, V-code, and E-code shapes	Fail
Valid ICD Diagnosis	Either ICD-9-CM or ICD-10-CM, for mixed columns	Fail
Valid ICD-10 Procedure	Seven characters, no I or O	Fail
Valid HCPCS J-Code	J followed by four digits	Warning
Valid CMS POS	Membership in the published CMS Place of Service set	Fail
Valid NCT	Eight digits, with or without the NCT prefix	Warning

Table 1: all twelve are Validity tests on a single column, and all belong to the Commercial Pharma generation set.

Four kinds of check, and the difference matters

The twelve sort into four groups. Knowing which group a test belongs to tells you how much a passing result is worth, which is the part that gets glossed over most often.

Two run arithmetic. Valid NPI and Valid DEA compute the check digit the specification defines and compare it against the digit sitting in the value. A failure here is not a matter of opinion.

Three cover the level an NDC column stores. Package, product, or unsegmented.

Six check shape only. The four ICD tests, Valid HCPCS J-Code and Valid NCT confirm a value is well-formed. They do not consult any code set, so a well-formed value that is not a real code will pass.

One checks membership in a published list. Valid CMS POS, and it is the only one. Hold on to that distinction.

Generic Regex vs. Healthcare Reality		
Every single critical failure hiding in your vendor feeds will pass a standard regex test.		
	Generic Pattern Match	Healthcare-Specific Check
Transposed NPI	PASS (Valid Shape)	FAIL (Broken Structure/Meaning)
Stripped 10-Digit NDC	PASS (Valid Shape)	FAIL (Broken Structure/Meaning)
Phantom POS Code (99)	PASS (Valid Shape)	FAIL (Broken Structure/Meaning)
ICD-10 containing 'O'	PASS (Valid Shape)	FAIL (Broken Structure/Meaning)

Arithmetic: the two check digits

An **NPI** is ten digits, and the tenth one is not data. It is a **Luhn** check digit computed over the first nine, with the **80840** prefix the spec requires folded into the sum. Transpose two adjacent digits and the sum stops landing on a multiple of ten, which is the entire reason the digit is there. Hand-keying and OCR produce transpositions, so the identifier was designed to catch them. A regex that counts to ten throws that design away.

PROBLEM 1: THE ARITHMETIC BLINDSPOT

A standard regex test counts to ten and stops. It cannot catch hand-keying or OCR transpositions. It checks the shape, but ignores the math.

NPIs do not fail because they are missing; they fail because they are mutated.



DEA registration numbers carry the same idea in a different arithmetic. The seventh digit has to equal the first, third and fifth digits added together, plus twice the sum of the second, fourth and sixth, mod ten. Valid NPI and Valid DEA run those calculations in SQL, on the landing table, on every run.

Level: which NDC your column actually stores

NDC is where most commercial pharma warehouses lose a day a quarter, and the reason matters, because people usually get it wrong.

The FDA assigns a ten-digit code in one of three segment layouts: 4-4-2, 5-3-2 or 5-4-1. Billing wants eleven digits in 5-4-2, which you get by padding the short segment with a leading zero. That works as long as the segments are still visible. Strip the hyphens and they are not. Take 1234567890 :

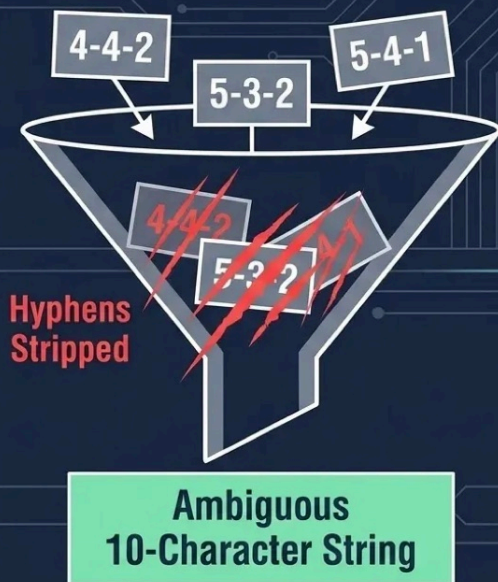
Layout	Segmented	Padded to 11
4-4-2	1234-5678-90	01234567890
5-3-2	12345-678-90	12345067890
5-4-1	12345-6789-0	12345678900

Table 2: three different products, the same ten characters, and nothing in the value to say which one you are holding.

PROBLEM 2: THE NDC AMBIGUITY TRAP

The FDA assigns 10-digit codes in three configurations (4-4-2, 5-3-2, 5-4-1). Billing demands 11 digits (5-4-2) for 867 files, padding the short segment with a leading zero.

This works until a warehouse strips the hyphens. When the hyphens vanish, the context is destroyed.



Be clear about what the three tests do and do not resolve. All three accept every FDA layout; none of them can tell you which layout a stripped value came from, and Valid NDC Any Form says so in its own usage notes. What separates them is the code level your column stores. Valid NDC takes the package-level three-segment form and the eleven-digit form your 867 files and claims data carry. It rejects a bare ten-digit string deliberately: that string is ambiguous, and treating it as valid is how the wrong product ends up in a report. Valid NDC Product takes the two-segment product-level form your product master and formulary files carry. Valid NDC Any Form takes any recognized form including bare digits, for the warehouse that strips separators as a convention and has decided to live with the ambiguity. Even that loosest one still catches the leading zero Excel ate, the `UNKNOWN` somebody typed into a required field, and the value that is somehow twelve characters long.

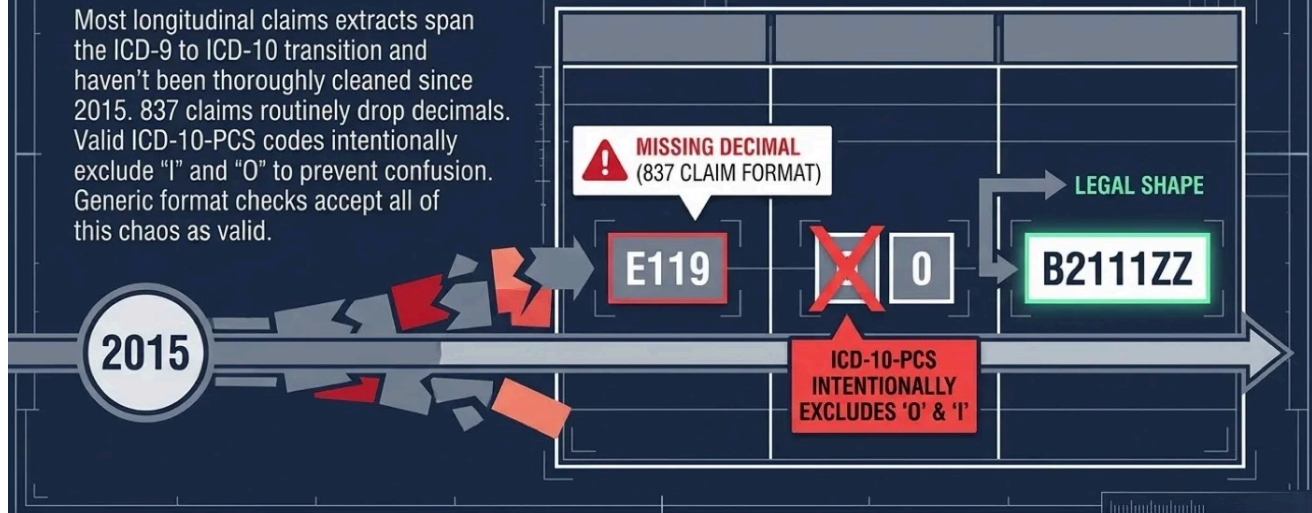
Shape: ICD, J-codes and trial numbers

Four tests cover ICD. Valid ICD-10 Diagnosis, Valid ICD-9 Diagnosis, and Valid ICD Diagnosis for the column that spans the transition and honestly holds both, which describes most longitudinal claims extracts nobody has cleaned since 2015. Valid ICD-10 Procedure checks the seven-character PCS shape, including that the code set leaves out `I` and `0` so nobody confuses them with `1` and `0`. The diagnosis tests accept dotted and undotted forms, because 837 claims drop the decimal and your vendor extract may or may not have put it back.

Valid HCPCS J-Code checks the J plus four digits that Part B buy-and-bill runs on, one slice of HCPCS Level II scoped to drugs. Valid NCT covers ClinicalTrials.gov identifiers on the medical affairs side, as NCT plus eight digits or eight bare digits.

PROBLEM 3: CONTEXTUAL DEGRADATION (CLAIMS & ICD)

Most longitudinal claims extracts span the ICD-9 to ICD-10 transition and haven't been thoroughly cleaned since 2015. 837 claims routinely drop decimals. Valid ICD-10-PCS codes intentionally exclude "I" and "O" to prevent confusion. Generic format checks accept all of this chaos as valid.

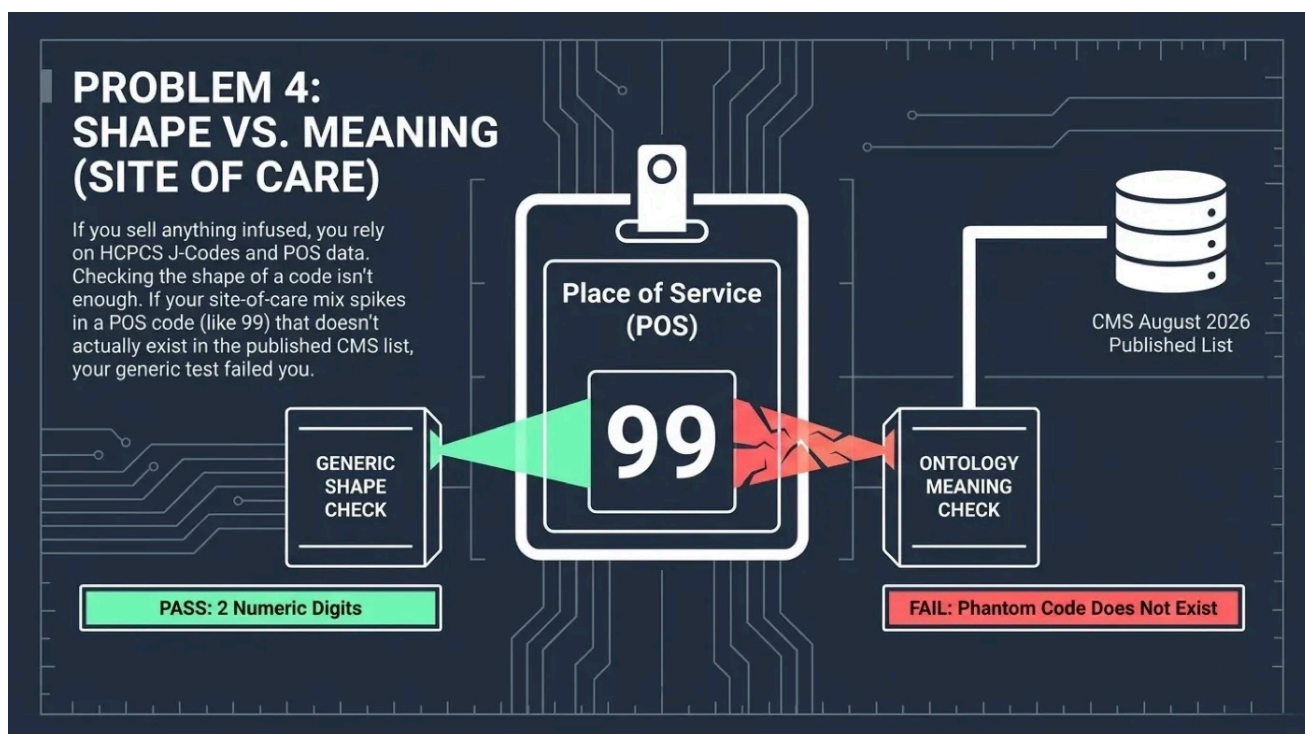


These six check that a value is well-formed. They do not check that it exists. `Z99.999` passes Valid ICD-10 Diagnosis, `000.00` passes Valid ICD-9 Diagnosis, `ZZZZZZ` passes Valid ICD-10 Procedure, and `J0001` passes Valid HCPCS J-Code, and not one of those is a real published code. That is a deliberate scope decision rather than a gap: the ICD-10-CM code set alone runs to more than 70,000 codes and revises annually, so carrying it inside a test would mean shipping a copy that silently goes stale.

Membership: the one test that checks a real list

Valid CMS POS is the exception, and the pack's genuine differentiator. CMS publishes Place of Service as a small fixed list, small enough that the test carries the actual list and checks membership against it. The copy in the pack is dated in its own usage notes: current as of August 2026, including codes 27, 58 and 66.

That means a Place of Service code the test rejects is worse than malformed. It does not exist. If your site-of-care mix suddenly shows growth in a POS code CMS never published, you would rather hear it from a test than from the slide.



You don't configure any of it

Profile the schema and the tests write themselves, using the same mechanic that [wrote data quality tests for 3 million NYC taxi rides in five minutes](#), pointed at identifiers instead of fare amounts.

Generation reads the column name and the profiled values together, and it needs both to agree. A column matching `%npi%` whose most frequent pattern is 10 digits gets Valid NPI. Unless the name says phone, fax, or contact. Unless profiling already tagged the values as a US phone number, which happens more than you'd like on vendor feeds where the HCP contact block sits three columns over.

The two-signal rule is not caution for its own sake. `B2111ZZ` is a legal ICD-10-PCS procedure code and a legal ICD-10-CM diagnosis code, so the shape cannot tell you which one you are holding, but the column name usually can.

The cost is that a column named `code_3` gets nothing. Add that one by hand, the same way you [shop the rest of the test catalog](#). That's the residue no profiler reaches, and on a commercial team [your best data quality rules live in someone else's head](#): the analyst who knows which specialty pharmacy reports units instead of scripts.

The specific test replaces the generic one

You already had Pattern Match, and on a column Valid NPI now covers it is noise. So when a Commercial Pharma test generates on a column, it removes the Pattern Match test that generated on that same column in the same run. Only that run's tests. Only unlocked ones. Anything you wrote by hand or locked stays exactly where it is. Before you confirm generation, the dialog tells you the most it will delete.

What these tests won't tell you

These tests check format, plus a check digit where the specification defines one, and with the single exception of Valid CMS POS they never consult a code set or call NPPES. A Luhn-valid NPI is therefore a number that could have been issued. It is not proof that one was, and it says nothing about whether the provider is still active or whether the NPI belongs to the prescriber your feed claims it does. A failure tells you something definite; a pass only narrows the possibilities.

The same limit runs down the list. A well-formed ICD-10 code from a category retired in 2019 passes. A DEA number carrying a registrant-type letter the DEA never assigned passes. Valid NCT accepts any eight digits, so on a column storing the numeric part alone, `20260815` passes.

None of this replaces Veeva Network or whatever you use for HCP mastering. MDM tells you what it could match. These tell you what arrived broken, per column, per feed, on the landing table, before a match rate has the chance to hide it. That distinction is the same one that makes [testing commercial pharma data harder than anyone tells you](#): the failures live between the feeds, and the feed itself never errors.

Empty and null values don't count as invalid anywhere in the pack. That's a Required Entry question and a different test. The format checks ignore trailing spaces, so an identifier sitting in a fixed-width `CHAR` column doesn't fail for its padding. The tests fold case first, since NCT and ICD codes come back lowercase from URLs and hand-built extracts constantly.

Why some fail and some only warn

Seven of the twelve default to Fail and five to Warning, and the line between them follows how much a failure actually proves.

A test defaults to Fail where a bad value can only be a bad value. Valid NPI and Valid DEA run a check digit, so a failure is arithmetic rather than judgement. Valid CMS POS checks membership in a list CMS publishes, so a failure is a code that does not exist. The four ICD tests check shapes the code systems define exactly, and a value outside those shapes is malformed whatever else it is.

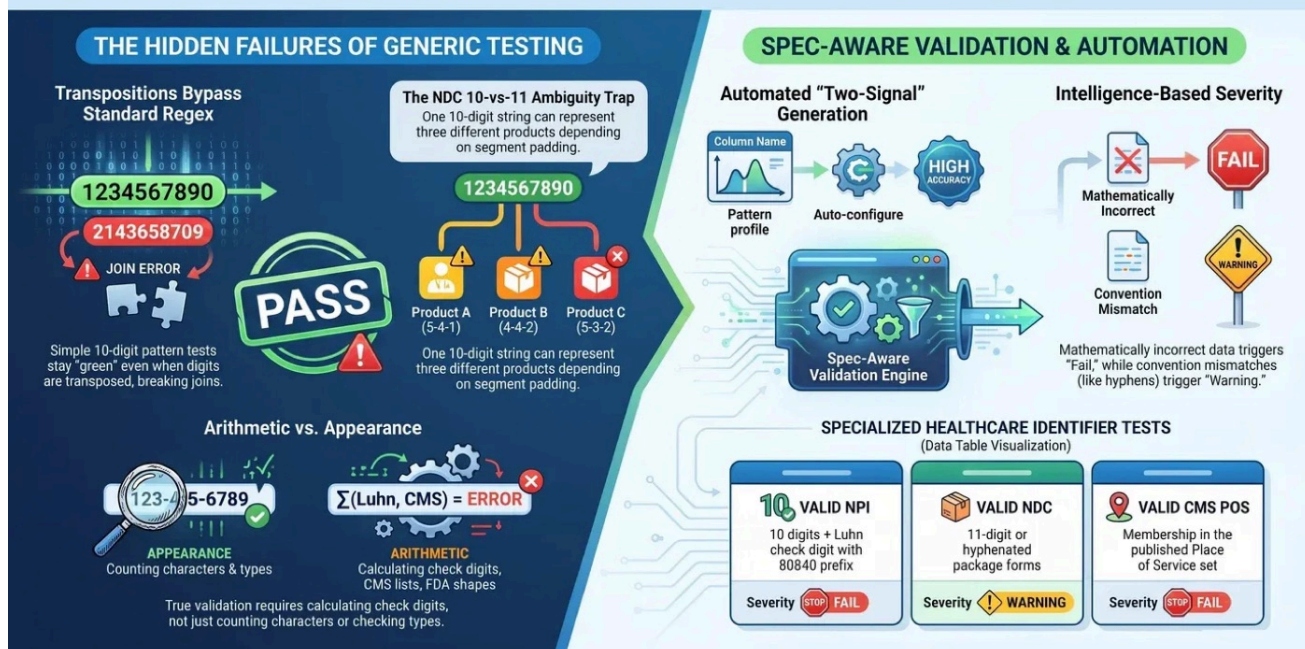
The rest default to Warning because a format mismatch is usually a convention difference. Your vendor sends NDCs with hyphens and your product master stores them without. Neither side is broken. Somebody just never agreed on a form. A J-code column that also carries the rest of HCPCS Level II lights up the same way, and that's a scoping decision rather than a data defect.

Change the default on any of them. Severity is set per test definition, so if unhyphenated NDCs really do break your gross-to-net, promote that one to Fail and let it stop the run.

The whole argument on one page

Something to send to the person who owns the feed:

Beyond the “False Green”: Precision Validation for Commercial Pharma Data



What a generic pattern test misses, on the left, and what the Commercial Pharma pack checks instead, on the right.

SECTION 2

The launch

1. The Launch Data Timeline and the 24 Reports
2. Celgene: Meeting the Product Launch Challenge with DataOps
3. Drug Launch Case Study: Amazing Efficiency Using DataOps
4. Hoarding cash before the FDA gate opens? Your data and analytics bill is the easiest line to cut.

The Launch Data Timeline and the 24 Reports

THE LAUNCH

The Launch Data Timeline and the 24 Reports

The DataOps Way to Commercial Pharma Data

DataKitchen

You get one chance at the field's trust. One wrong territory number in the first month and the reps go back to running on gut. After that the platform you paid for is furniture.

That is the stake behind every launch data platform, and it is why the platform is judged on two things only. Does it exist early enough to be trusted? And can it carry the two dozen recurring reports a commercial team runs on? Everything else is detail. This chapter lays out the calendar first, then the inventory.

Why the window is fixed

The first six to twelve months set the trajectory a brand rides for the rest of its life. A solid early ramp puts a medicine on track to hit its numbers. A weak launch costs revenue nobody recovers.

So the data has to be right before it matters, which means the platform has to be running before anyone is watching it. Field reporting live six months before launch day is the only way we have found to get the numbers trusted on day one. The field learns the reports, argues with them, finds the edge cases, and settles in while a wrong number still costs nothing.

Twelve to 24 months out is the cheapest and calmest time to start. If the product is already selling, the timing works differently: you take the existing operation over feed by feed, and there is no window to miss.

Build, run, transfer

We run a launch in three phases with clear roles. We hold the engineering lane. Your analysts hold the business lane. The platform transfers when you say so.

Build runs from roughly month minus 18 to month minus one. This is the commercial data platform your launch needs: star schemas your analysts can query, the AI-ready context layer beside them, tests on every table, and observability across every pipeline. It ends with a production platform six months before launch day, with field reporting in front of the reps.

Run covers launch day through year one. We sit in your standup. New datasets ship in days. Breakage gets fixed before the field sees it. Your analysts spend their week on analysis rather than reconciling the IQVIA refresh against claims, and your AI tools answer real questions because the data underneath them is real. The measures of success are plain: the launch hit its numbers, the analyst queue stayed under a week, and errors were caught upstream of the field.

Transfer happens whenever you say. You own the platform from day one, so the handover is a normal day rather than an event. It goes to whoever you name: your team, Accenture, or your offshore center. Eisai moved one of our platforms to Accenture cleanly, because every line of code and every test already lived in their environment. There was nothing to extract.

If the launch slips, the engagement scales down on 30 days' notice, with no penalty for an FDA delay you did not choose. That matters more than it used to. Roughly 3,500 staff left the FDA after the 2025 cuts, and biotechs are missing meetings and pushing trials. A fixed multi-year commitment is the wrong shape for a date you do not control.

You are not waiting two quarters for a design document

The milestones come fast on purpose. Analysts should be giving feedback while the platform is still cheap to change.

Week four: the first star schema. A fast-start star in a data mart on your highest-priority feed, with tests on it from the first build.

Week six: the first AI context layer. Schema and grain written down, business definitions for equalized prescriptions and payer hierarchy, and validated example queries, so your AI tool answers instead of guessing.

Month minus six: field reporting live. The full launch report set in production and in front of the field.

Launch day: the first year begins. The trajectory gets set now.

Ongoing: new datasets in days. A new business question that needs a large new dataset gets a fast-start star, questions back to the supplier, tests, and analyst feedback across a few sprints rather than a quarter.

Through all of it the mechanics stay boring. Daily scrum and ticket triage in Jira. A weekly status meeting with your analytics lead covering work completed, in progress, and planned. A quarterly business review with leadership on timelines and on the metrics for bugs, tests, and datasets. Documentation in a shared wiki. Time tracked and reported.

What these teams actually looked like

The case for small teams is not theoretical.

At Karuna, one and a half engineers built and ran the entire Cobenfy launch data infrastructure over two years, covering 50 integrated datasets. Most pharma companies staff that work with ten engineers.

At Celgene, seven of our engineers supported US sales across every brand, about \$10 billion in sales, with ten to 12 analysts covering hundreds of datasets and no missed SLAs. Only 18 percent of that engineering time went to data loading. The rest went to new analyst work rather than plumbing, which is the whole point of automating the loads.

The 24 reports a launch analytics team is on the hook for

A data team of one or two dozen people serves hundreds of sales and marketing users, and each of those users has a territory or a campaign. Their requests resolve into the reports below. Every row needs data from the syndicated and internal feeds a commercial team buys.

First, whose job is which. These are your team's reports, not ours. Your analysts own the analytics, the dashboards, and the business logic inside them, and they should. The data layer every row depends on is what has to be built and tested, so the numbers hold when someone questions them. Incentive compensation design, territory alignment, call planning, and brand strategy are not data engineering work and stay with the business.

Performance and forecast

The numbers leadership sees first, and the ones a silent vendor restatement quietly rewrites underneath you.

Report	What it is	Cadence	Who reads it	Key metrics
Forecast tracking	National brand performance against the current forecast, built on national prescription audit and forecast metrics.	Weekly	Senior management, brand team	TRx, NRx, units
Forecast creation	Developing the forecast itself from analogs, historical trends, population facts, and projected events.	Quarterly	Senior management, brand team, finance	TRx, NRx, units
Launch tracking	The executive summary that pulls the key point out of every more detailed report below.	Weekly	Senior management, brand team, home office	TRx, NRx, cohorts, geographic differences, deciles, specialties, targets, activity vs plan, payer performance, top prescribers

Report	What it is	Cadence	Who reads it	Key metrics
Claims reporting	Claim counts per product over time, from syndicated claims data.	Weekly	Senior management, brand team, field	Claim volume and mix

Brand, promotion, and ROI

Where sub-national detail matters, and where attribution windows shift the answer when a channel feed lands late.

Report	What it is	Cadence	Who reads it	Key metrics
Brand team report	Critical sub-national topics in depth, joining sub-national volume, prescriber reference, alignment, targeting, promotional and field activity.	Monthly	Brand team	TRx, NRx, cohorts, deciles, specialties, targets, payer performance, promotional execution and results
Brand planning support	Standing up the analysis behind annual brand planning, including drivers analysis.	A few months, annually	Senior management, brand team	Multiple
Promotional mix and ROI	Lift analysis and return calculation across the major promotional efforts.	Monthly	Brand team	TRx, NRx, units, financial return
Direct-to-consumer	DTC geographic coverage and the lift analysis on top of it, keyed to current alignment.	Monthly	Brand team	TRx, NRx, units
Non-personal promotion	Lift and ROI across digital channels, joining promotional data to sub-national volume and field activity.	Weekly	Brand team	Clicks, views, reads, lift

Field execution and incentive comp

The reports reps and sales operations live in, and the ones where a territory realignment or a duplicated prescriber costs real money.

Report	What it is	Cadence	Who reads it	Key metrics
Field activity	Rep activity against planned activity, joining prescriber reference, alignment, targeting and activity data.	Weekly	Field	Calls, samples
Field performance	Brand performance by geographic alignment. This is the report a realignment breaks while every underlying value stays valid.	Weekly	Field	TRx, NRx, calls, samples
Pre-call planning	Physician-level views that integrate several datasets into one page a rep reads before a call.	Weekly	Field	Multiple metrics by physician
Incentive compensation	Ranking each rep on performance and business logic. A duplicated prescriber splits credit here and the comp run pays the wrong people.	Weekly	Field, sales operations	IC metrics, rank
Plan of action and sales meetings	The analysis behind national sales meeting presentations, sessions, and materials.	At meetings	Field	Field activity and performance
Target list development	Identifying the most promising existing and potential prescribers from prescriber reference, alignment, and sub-national data.	Periodically	Sales operations	Scores by physician, zip-to-territory

Report	What it is	Cadence	Who reads it	Key metrics
Sales force logistics	The periodic data work that keeps the field functioning between the headline reports.	Periodically	Sales operations	Multiple

Access, payer, and patient

Where a specialty brand actually lives, and where a half-loaded dispense feed hides real patients during the weeks that matter most.

Report	What it is	Cadence	Who reads it	Key metrics
Payer and plan	Performance across major model groups, minor model groups, and individual payers, which usually needs a bridge file to join.	Monthly	Brand team	TRx, NRx, units, formulary status impact
Stocking	Which pharmacies carry the brand, from syndicated stocking data.	Weekly	Brand team	Pharmacies stocked
Specialty pharmacy	Stocking and sales across the specialty network, stitched from a dozen or more pharmacy feeds that disagree on what a date means.	Periodically	Senior management, brand team	Dispense, referrals, time to fill
Source of business, patient level	Adherence, refills, and switching from longitudinal patient and source-of-business data.	Monthly	Senior management, brand team	Adherence, persistence, discontinuation

Finance and compliance

The reports that have to tie out, where "roughly right" is not an available answer.

Report	What it is	Cadence	Who reads it	Key metrics
Finance reporting	Accrued financial metrics from finance data.	Monthly	Senior management, brand team, finance	Multiple financial metrics
Demand-based P&L	TRx, SKU, pricing, and contracting data combined into a demand view of profitability.	Monthly	Senior management, brand team, finance	Multiple financial metrics
REMS	Use within the certified components of the delivery chain, from REMS-specific datasets.	Daily, weekly, or monthly by program	Brand team	REMS program metrics

The queue nobody schedules

The work that decides whether your analytics team is trusted, and the first thing to collapse when the engineering layer underneath is weak.

Report	What it is	Cadence	Who reads it	Key metrics
Analytic questions	Answering questions about the launch and sub-national market dynamics quickly enough to still matter, across whatever data the question touches.	On tap	Extended sales and brand team	Multiple

That last row is the one to watch. Twenty-three reports can be on time and correct, and the team is still not trusted if a brand lead's Tuesday question gets answered on Friday. The queue nobody schedules is where the data layer either holds up or gives way.

Celgene: Meeting the Product Launch Challenge with DataOps

Read the original article at: <https://datakitchen.io/blog/meeting-the-product-launch-challenge-with-dataops/>



Bringing a drug to market costs \$2.6 billion, according to the Tufts Center for the Study of Drug Development. You get one launch. The first six to twelve months decide how fast the product grows, and how fast it grows in that window sets its revenue for the rest of its life.

That is the pressure the commercial analytics team at Celgene, a Fortune 500 pharmaceutical company later acquired by Bristol Myers Squibb, was under for the Otezla launch.

Rajesh Gill, Associate Director of Commercial Insights at Celgene, tells this story himself: [The Celgene Story: Building a \\$1B Product Launch Success with DataOps](#)

What the platform had to carry

The commercial analytics platform ran on Redshift feeding Tableau Online. More than 1,000 dashboards served hundreds of sales people, plus marketing and executives.

Everything upstream arrived on someone else's schedule: syndicated data, sales data, Rx claims, specialty pharmacy feeds, NPP events and campaigns, sales alignments, product hierarchies, and specialty mappings. A launch does not wait for any of them to be clean.

What DataOps changed

On the DataKitchen platform, the team integrated hundreds of data sets into a mastered, unified star schema, covered by more than **20,000 automated tests**. The platform absorbed **over 100 schema and data changes per week** with very few errors and very few missed SLAs, at a low total yearly cost across hardware, hosting, software, and staffing.

Those two numbers belong together. Absorbing a hundred upstream changes a week is only survivable if tests catch what the changes broke before a sales rep opens a dashboard.

The engineering side: scrappy star, then solid star

A new large data set moved through six steps:

1. Build a scrappy star in a data mart.
2. Send questions back to the data supplier, keeping analysts in the loop.
3. Add data tests.
4. Share the star with the analyst team for feedback.
5. Iterate over several Agile sprints.
6. Release a solid star.

The point is the order. The scrappy star exists so the analysts have something real to react to while the questions to the supplier are still open.

The analyst side, running in parallel

Analysts did not wait for a finished star. They built scrappy dashboards, fed corrections back to data engineering, and showed early dashboards to users. Build and design sessions were active, with as many changes made live as possible.

Then they published production dashboards at **70 percent done**. Feedback from real users on a partial dashboard is worth more than a longer wait for a complete one.

What the team owed the business every week

The analytics team carried three kinds of work at once, and the mix is the part people underestimate when they plan a launch:

- **A standing production deliverable.** The weekly launch and performance tracker, which goes out whether or not the upstream feeds cooperated that week.
- **Ad hoc answers.** A brand leader asks a question on Tuesday that nobody anticipated on Monday.
- **Models.** Resource allocation and prediction, which need the same data to be stable enough to model on.

They instrumented themselves as well as the data, tracking time spent, tickets opened and closed, query performance, which data sources had arrived, and the data errors with their detail. A launch team that cannot say where its week went cannot defend its headcount, and a team that cannot show its error rate falling has only opinion to offer when someone asks whether the numbers can be trusted.

Span of control

The framing Rajesh used for all of this was span of control, and it is the most portable idea in the story.

An analytics leader in a launch sits between data sources and IT systems on one side and marketing, sales, and customers on the other. Plenty of that surface is outside their authority. The question is which parts have to be inside it. His answer: the data, the analytic database, the people, the tools, and the processes that deliver value. Five areas, held deliberately: agile team management, the DataOps platform itself, the data engineers with their tools and database, the analysts with theirs, and data operations.

The reason to insist on that is the pace. When a syndicated feed changes shape the week before a field deadline, a team that has to raise a ticket with a group that reports elsewhere has already missed the deadline. The same fix inside your span of control is a sprint task.

The guiding principles were equally blunt, and one of them explains the rest: no plan survives first contact with reality. Rely on expertise as much as experience. Make sure every subteam understands the brand strategy, not just their piece of it. Assume there will be issues. Build the pathway for rapid decisions before you need it.

The five substitutions

The technology mattered less than the mindset shift underneath it. Five substitutions:

- Change fear becomes change velocity.
- Manual operations become automated operations.
- Hope for quality becomes integrated quality.
- Hero mentality becomes repeatable processes.
- Perfection becomes 70 percent right the first time.

The last one is the hardest to accept and the one that made the rest work.

Drug Launch Case Study: Amazing Efficiency Using DataOps

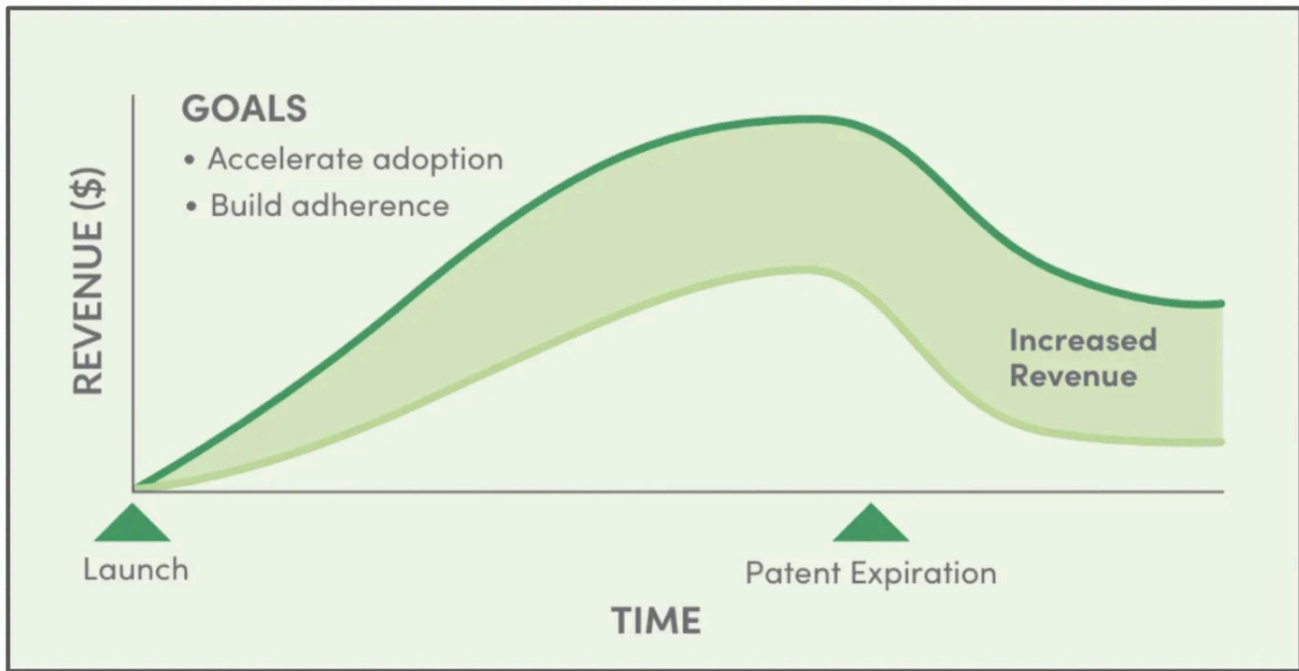
Read the original article at: <https://datakitchen.io/blog/drug-launch-case-study-amazing-efficiency-dataops/>



A Drug Launch Case Study in the Amazing Efficiency of a Data Team Using DataOps

When launching a groundbreaking pharmaceutical product, the stakes and the rewards couldn't be higher. This blog dives into the remarkable journey of a data team that achieved unparalleled efficiency using DataOps principles and software that transformed their analytics and data teams into a hyper-efficient powerhouse. This team built data assets with best-in-class productivity and quality through an iterative, automated approach. We will cover four parts: establishing the infrastructure, getting the data, iterating and automating, and using small, empowered teams.

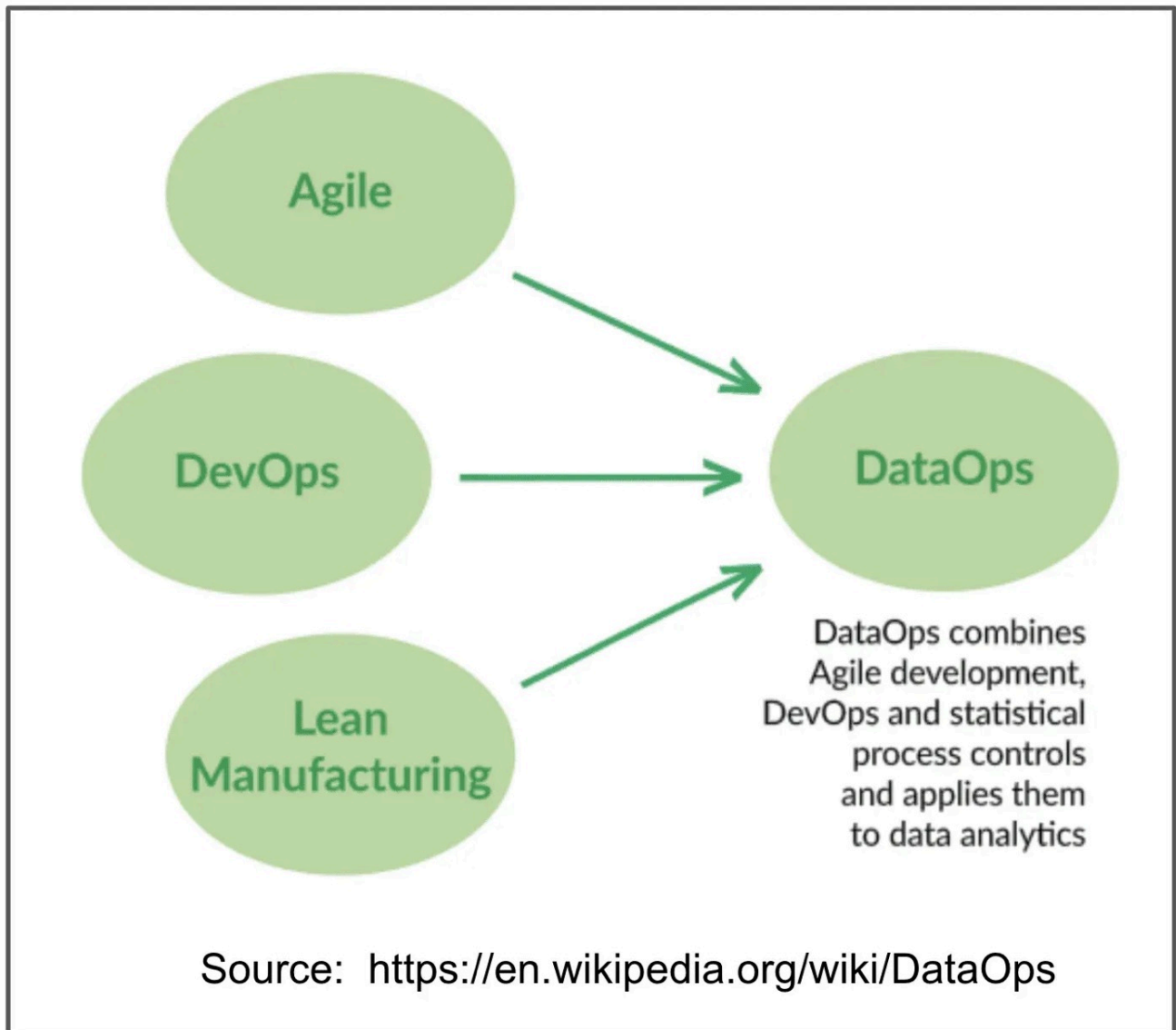
Since the speed at which a product grows during a launch determines the overall lifetime revenue (see graphic below), this team knew it had just one chance for a successful launch.



Their efforts culminated in more than just a successful drug launch, it fueled the acquisition of their company for billions of dollars. The numbers speak for themselves: working towards the launch, an average of 1.5 data engineers delivered over 100 lines of code and 1.5 data quality tests every day to support a cast of analysts and customers. Whether you're curious about the infrastructure, the productivity, or the mindset that made this success story possible, you'll want to keep reading to uncover the steps that turned a vision into a multi-billion-dollar reality.

Guiding Principles

The foundation of the success relied on DataOps principles. Applying DataOps , an Agile, automation-focused approach to data engineering , ensured a dynamic feedback loop where data delivery and insights could evolve rapidly.



This agility translated into faster, more reliable data for stakeholders and empowered the team to iterate. The focus was on delivering value.

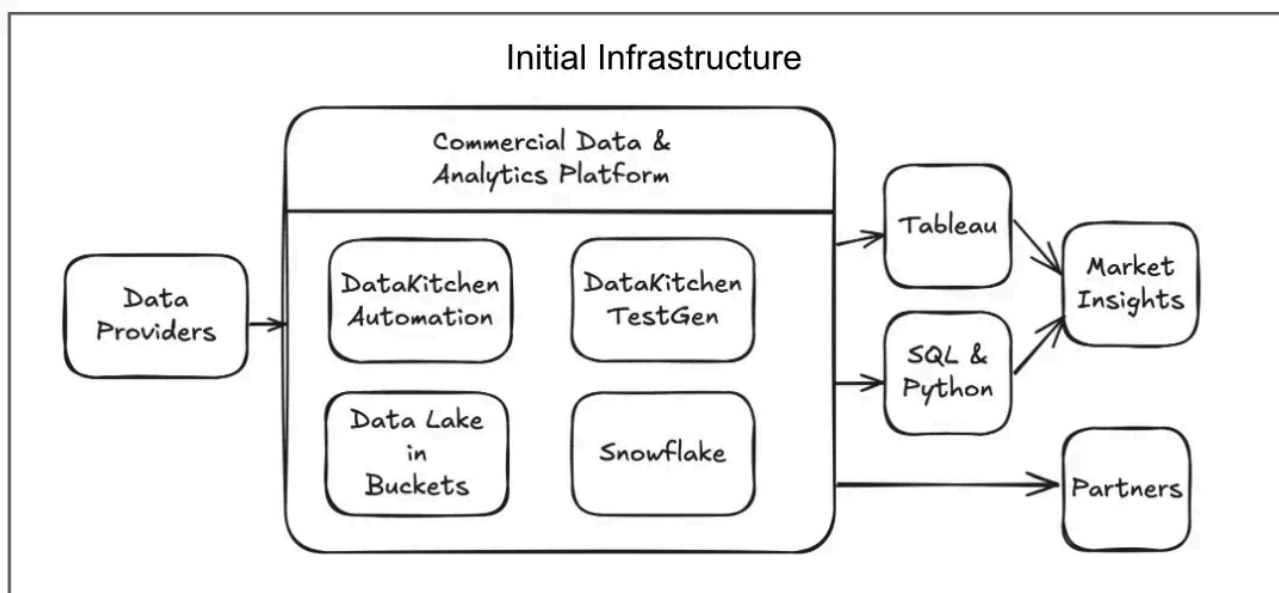
The project showed that smaller, empowered teams achieve higher impact than larger ones. These small, cross-functional teams ensured that members were deeply involved in the project operations, the technical setup, and the feedback cycle, leading to fewer delays, fewer bottlenecks, and faster decision-making. The company focused on delivering small increments of customer value - data sets, reports, and other items as their guiding principle. They selected best-in-class partners, tools, and vendors with that guiding principle in mind. Small, manageable increments marked the project's delivery cadence. Starting simply and iterating quickly gave the team time to build foundational processes before adding complexity and scaling. Finally, embracing a 'don't be a hero, automate' culture, the project emphasized owning and improving their work process rather than relying on extraordinary individual effort, ensuring sustainable progress and a stable, repeatable workflow.

For the data engineering team, these principles focus on building data assets that analysts trust, can control, and easily understand.

Establish the Infrastructure

The project began with establishing the data infrastructure. They opted for Snowflake, a cloud-native data platform ideal for SQL-based analysis. AWS Redshift, GCP Big Query, or Azure Synapse work well, too. The team landed the data in a Data Lake implemented with cloud storage buckets and then loaded into Snowflake, enabling fast access and smooth integrations with analytical tools. Once the foundation was set, the team brought in the first one-time data cut, an essential step for early-stage analysis that established the company's understanding of available insights. This cloud resource setup ensured the team could easily spin up environments, process large volumes of data, and adjust configurations quickly.

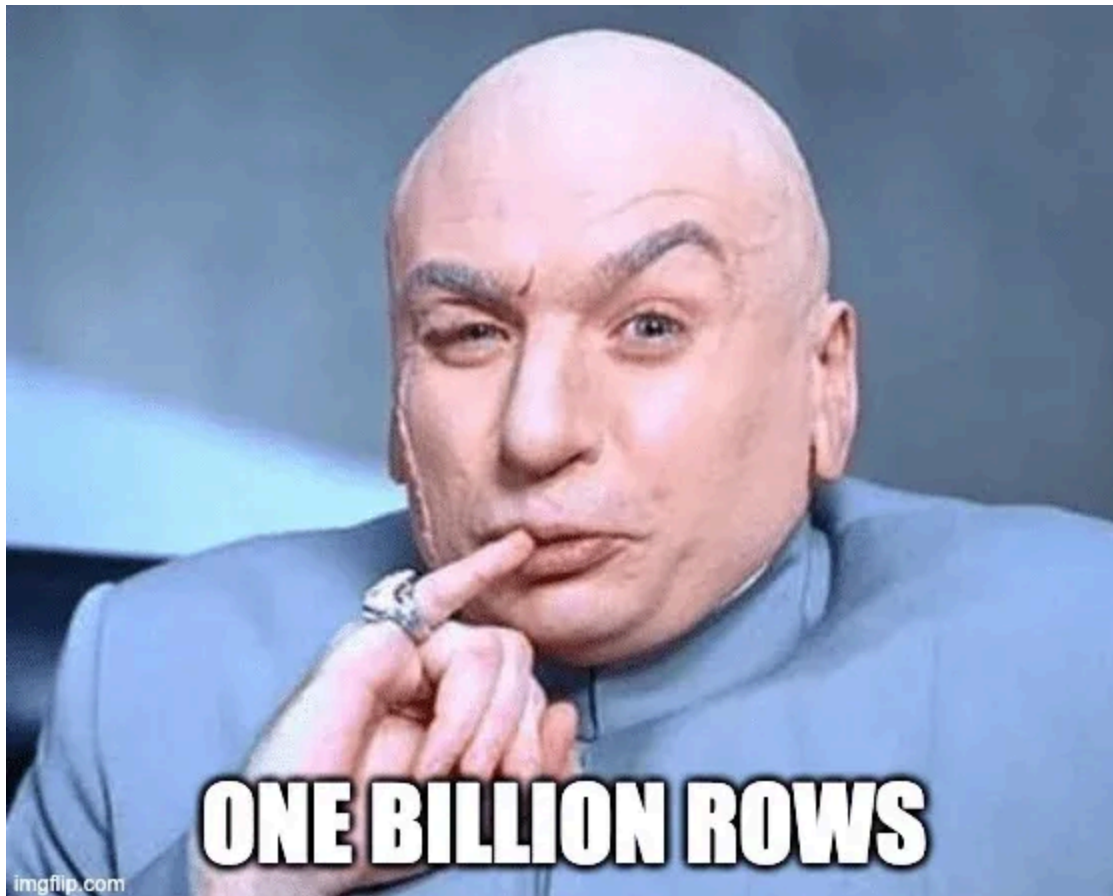
It is necessary to have more than a data lake and a database. A software system where processes can be developed and shared is required. The team used DataKitchen's DataOps Automation Software, which provided one place to collaborate and orchestrate source code, data quality, and deliver features into production. The following diagram shows what the initial infrastructure looked like.



Get the Data

Securing data was another critical phase. A project of this scale required high-quality, historical data that could offer insights into market behavior over time. The company's analysts, guided by DataKitchen engineers, worked with over a decade of claims data, which consisted of tens of billions of rows. DataKitchen loaded this data and implemented data tests to ensure integrity and data quality via statistical process control (SPC) from day one. By publishing a data dictionary to a wiki, analysts could quickly familiarize themselves with the data sets, providing a common language and understanding of the

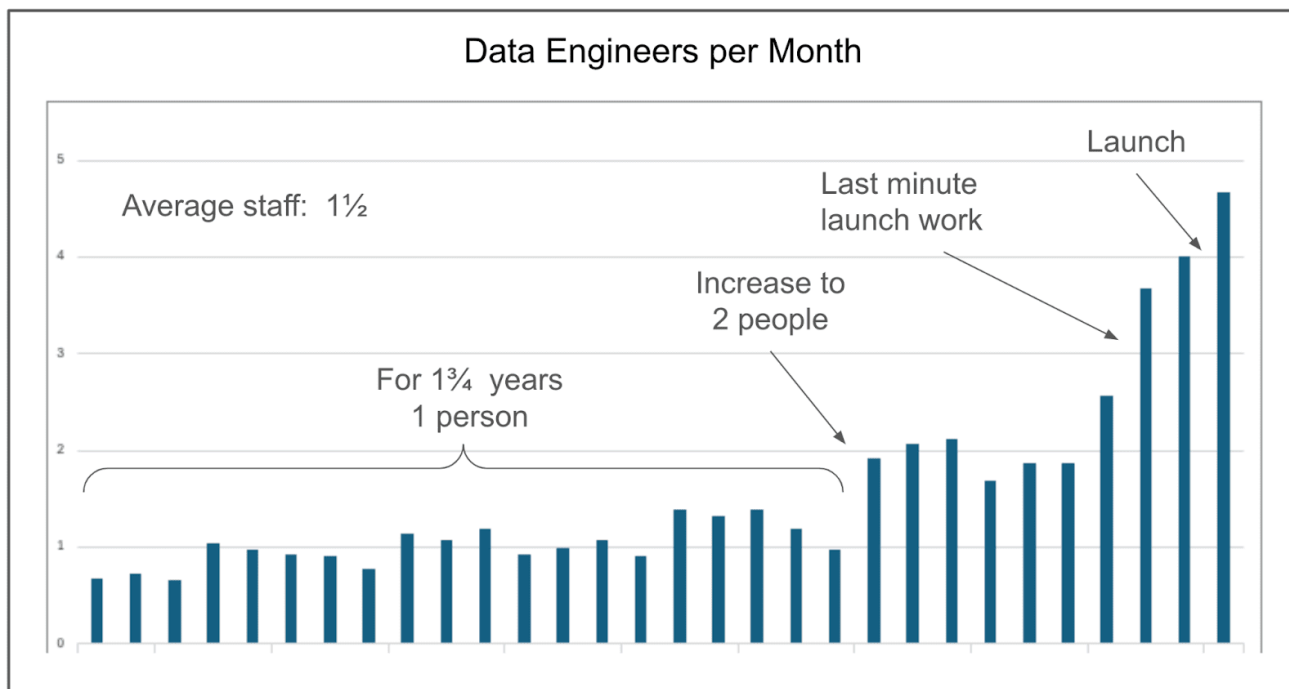
data, a valuable resource for new and seasoned team members. This centralization reduced confusion and established data standards early on, setting up a solid foundation for future analysis and a single source of truth.



Iterate and Automate to Analyze the Market and Establish Goals

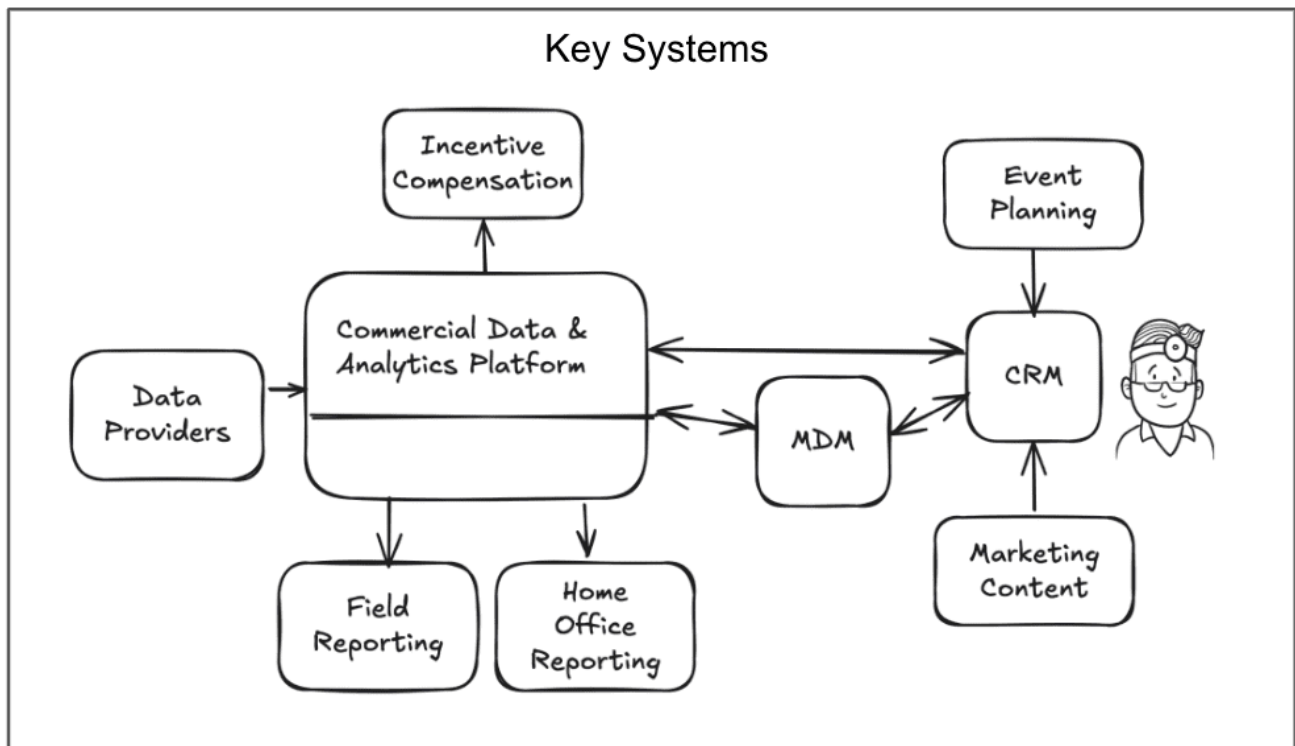
The journey continued with data iterations. The company experimented with various data from sources such as IQVIA and Symphony Health, both well-known providers of pharmaceutical sales data, but they didn't stop there. They explored claims data options from Komodo Health, MMIT, and Fingertip, evaluating the strengths of each set. The diverse range of data on NRx, TRx, sales force alignment, and zip code-to-territory mappings. EDI data, including 852, 844, and 867, added further depth to the project, enabling the team to glean additional insights. This variety was essential because the company's data needs were both complex and nuanced. Even with a highly skilled team, forecasting the unique dynamics of a new drug required continuous validation of assumptions against these different data sources, ensuring a robust, well-rounded analytical approach.

The company was able to do this exploration cost-effectively. One data engineer was able to support the project for the first 1¼ years due to the best practices employed, including DataOps principles and the DataKitchen software to support the process. It wasn't until nine months before launch that staff was increased to two people.



evaluated Constant Contact, Hubspot, and Salesforce Marketing Cloud for customer relationship management. Beghou, ZS, and KMK were contenders for call planning and salesforce design, while Hybrid Health and CVENT were considered for marketing content and event planning. Furthermore, an incentive compensation system was implemented to manage reward structures based on the data in Snowflake that was transformed and quality checked by DataKitchen software. The collaboration of these systems established a comprehensive digital ecosystem for the company's commercial operations, ensuring every aspect of the marketing and sales journey was data-informed and optimized.

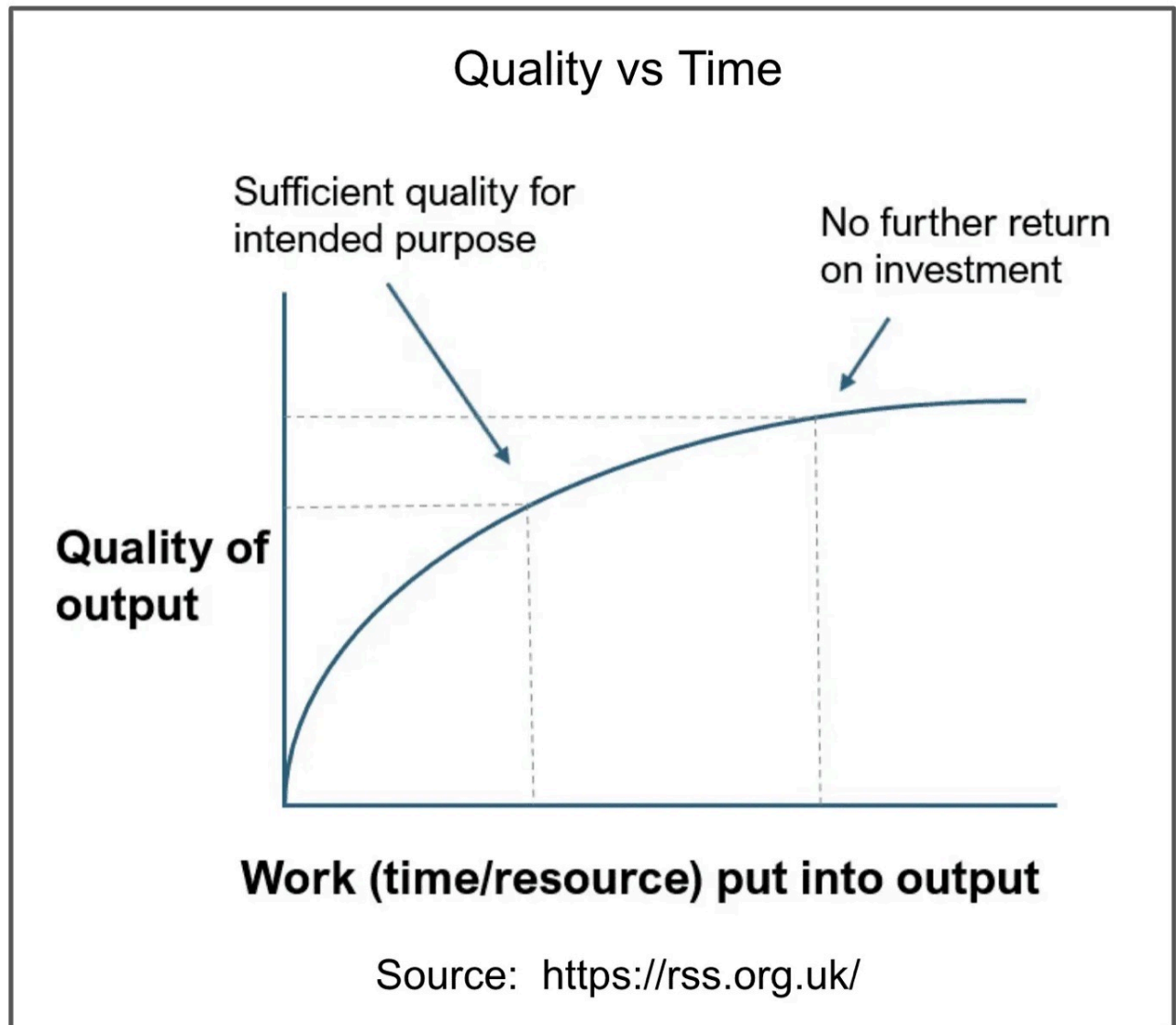
The following diagram shows the relationships between the key systems.



Start Early

The project culture fostered a balanced approach to time and expectations. Ample time to complete tasks reduced mistakes, allowed thorough documentation, testing, and automation, and ultimately enhanced the quality of the entire operation. Thousands of data tests were developed, and this extensive testing infrastructure enabled the deployment of analytics updates without fear of regression. Mistakes were seen as learning opportunities, and the anti-hero culture meant no individual bore the weight of project success. Systems were set up to allow smooth transitions and reduce the need for overreliance on individual effort. The long-term, sustainable success of the project reflected a “marathon, not a sprint” mentality, with strong systems supporting reliable execution over the project lifecycle.

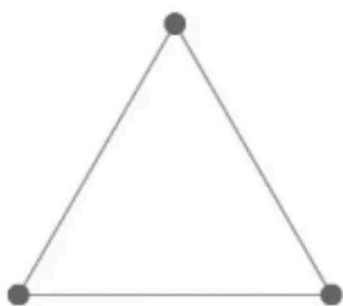
A paper by the Royal Statistical Society shows that as more time is given to a task, quality goes up, and after a point, you hit diminishing returns. The managers of this project were able to hit that sweet spot. See the graph below.



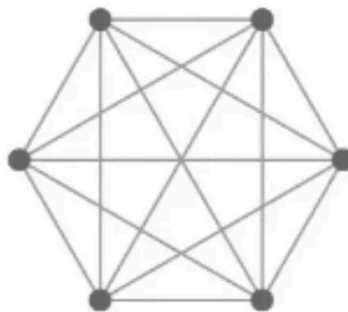
Use Small Empowered Data Teams

A key success factor was the emphasis on small, empowered teams. Communication was streamlined, which reduced the need for frequent meetings and excessive coordination. Team members knew who to approach for specific tasks or clarifications, enhancing productivity and accountability. This environment encouraged ownership of tasks and was conducive to proactive problem-solving. Remember that communication increases exponentially, $(N^2 - N)/2$, with the number of individuals involved.

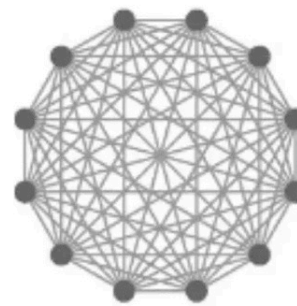
Communication is easier with small empowered teams



3 people, 3 lines



6 people, 15 lines



12 people, 66 lines

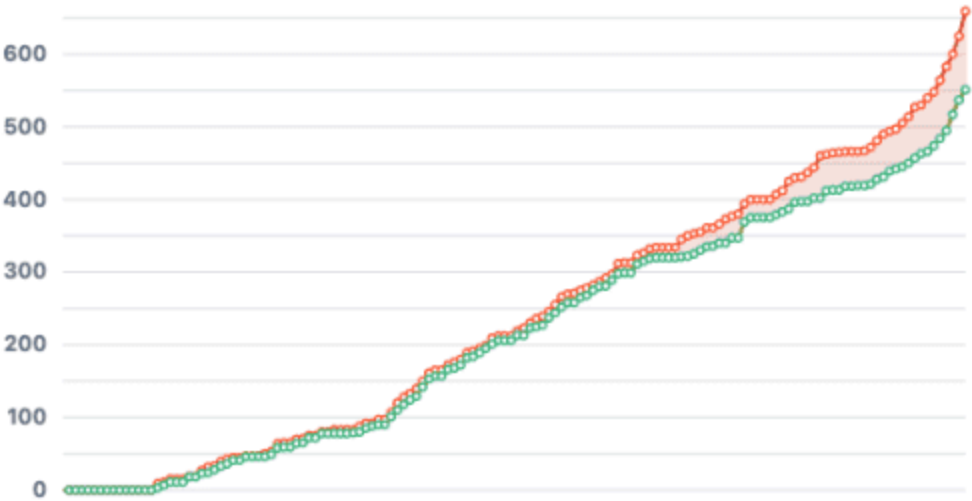
<https://deviq.com/laws/brooks-law>

Results

In the end, the company team's data infrastructure and analytics environment were well-prepared for the launch, with key functions such as field reporting ready six months before the expected launch. Reports for incentive compensation, claims, Rx data, payer plans, marketing automation, and more were prepared and fully functional, positioning the company for success. The ability to deploy reliable data pipelines across various analytics functions instilled confidence and paved the way for efficient, effective operations within the broader organization.

The team was able to achieve excellent technical productivity results.

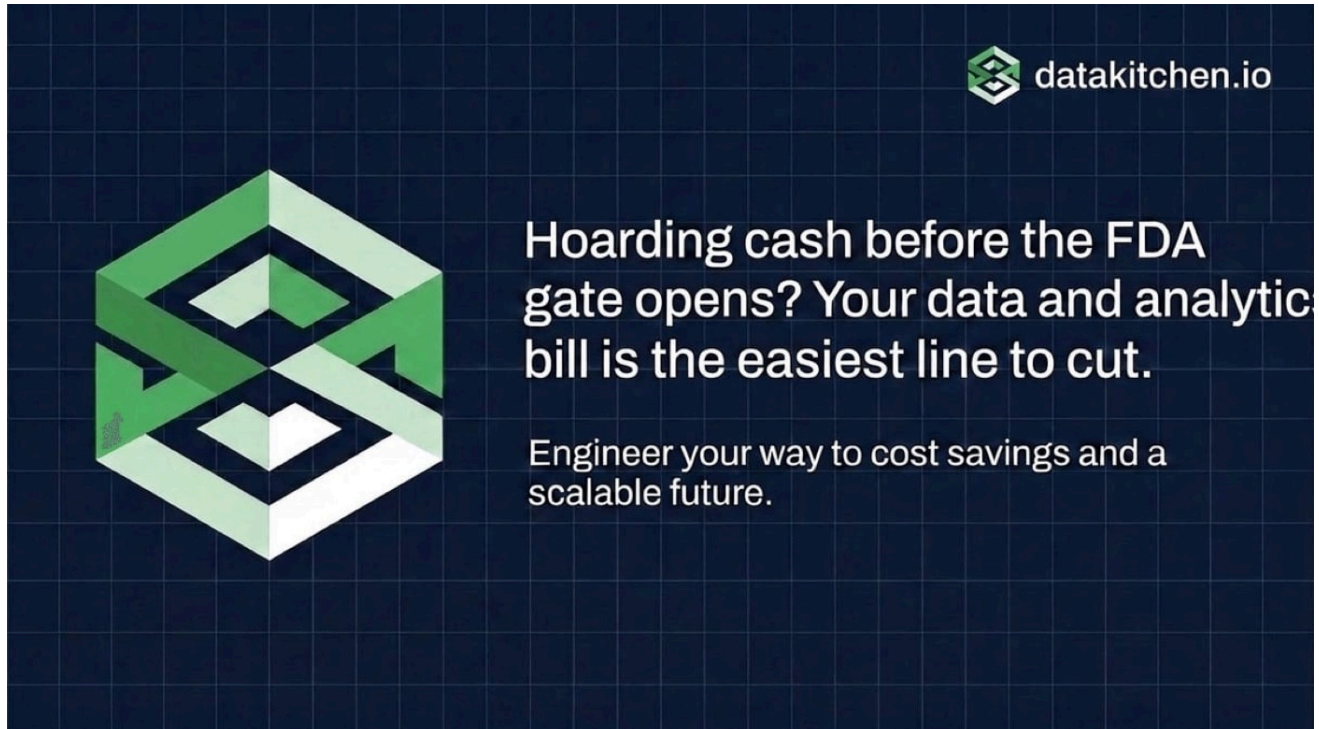
Tickets Resolved Over Time (green line)



THE LAUNCH

Hoarding cash before the FDA gate opens? Your data and analytics bill is the easiest line to cut.

Read the original article at: <https://datakitchen.io/blog/small-pharma-hoarding-cash-before-fda-gate-opens/>



You're a small pharma company. Maybe you've got one or two assets, maybe you're a few quarters out from your first commercial launch, maybe you're staring down a PDUFA date and trying to figure out how thin you can run the burn until then.

The macro picture is not friendly.

The FDA is slower than it used to be. After the 2025 cuts, roughly 3,500 staff left the agency, and biotechs are missing meetings, missing deadlines, and pushing trials. If your runway assumes a clean approval timeline, you're underwriting a risk you can't control.

THE MACRO REALITY IS FORCING RADICAL EFFICIENCY

THE FDA GATE

Slower approvals. 3,500 staff left the agency after 2025 cuts. Missing deadlines and pushing trials are un-underwriteable risks.

THE MANDATE: SMALL PHARMA CASH RUNWAY

Your job for the next 12-24 months: Keep the powder dry. Do not burn runway on consulting retainers or bloated data stacks.

THE PRICING SQUEEZE

\$529 billion in projected White House MFN savings. Large pharma is cutting \$32 billion in expenses by 2030 to adapt.

The pricing picture is worse. The White House is projecting **\$529 billion in savings over ten years** from most-favored-nation pricing. The administration says it expects similar agreements with most manufacturers of sole-source brand-name drugs. Seventeen big pharma firms have signed. Richard Pops, now chair of the Midsized Biotech Alliance of America at Alkermes, has said a sloppy MFN law could “inadvertently take out” companies like his. That means companies like yours, too.

The ZS **Pharma outlook for 2026** names the squeeze. Lower prices. Higher tariffs. Less NIH and FDA capacity. Tighter VC. The big players need to cut \$32 billion in expenses by 2030. The small players don’t have that kind of slack.

So your job for the next 12 to 24 months is to stay alive without going nuts. Keep the powder dry. Don’t burn the runway on a CRO bill, a consulting retainer, or a five-figure-a-month data stack that produces decks nobody reads.

Two problems are hitting at the same time

Your CEO wants AI fast. Your CCO wants the launch to hit its numbers. Your data team has never launched a brand before. None of that works without two things happening at once.

Data engineering has to ship fast with a tiny team. The data underneath your AI tools has to be ready, or **the chatbot guesses and is confidently wrong**.

Most vendors do one or the other. ZS and Beghou provide analytics consulting, billed at \$300 per analyst per hour. Accenture and Deloitte each have a 30-person data engineering team. The Snowflake and Databricks partners know the platform. AI-first vendors sell the chatbot on top. Nobody does both halves.

Move from extractive consulting to AI engineering

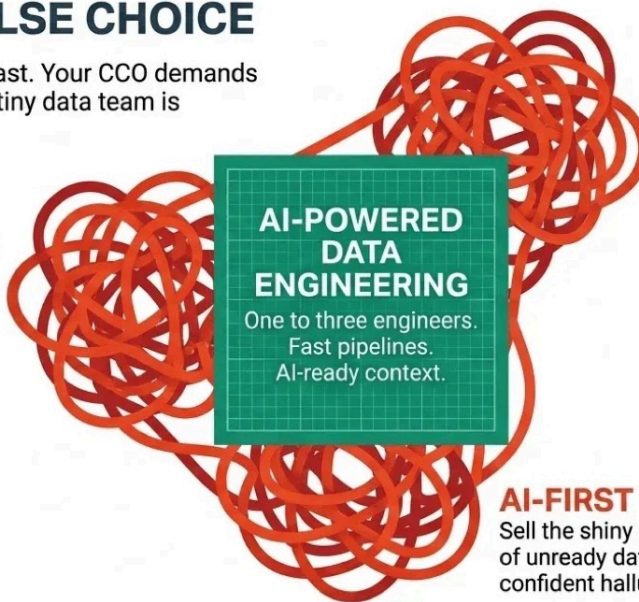
We use AI to do data engineering 10x faster. Our engineers run Claude Code inside our **DataOps + FITT + Data Testing framework**. The AI writes the SQL transforms and the data quality tests. Our **DataOps Observability** tool watches the pipelines and triages failures. Our engineers focus on the parts AI can't do: deciding what to build, designing the right data shape, and talking to your team.

This is why 1.5 of our engineers built the entire Cobenfy launch data platform at Karuna over two years. Most pharmas would staff that work with 10 engineers. Field reporting was ready six months before launch.

THE LAUNCH COLLISION AND THE FALSE CHOICE

Your CEO demands AI fast. Your CCO demands a flawless launch. Your tiny data team is caught in the middle.

ANALYTICS CONSULTANTS
e.g., ZS, Beghou.
Billed at \$300/hour.
They rent you bodies, not automated systems.



BIG TECH INTEGRATORS
e.g., Accenture, Deloitte.
Bring 30-person data engineering teams to build bloated infrastructure.

AI-FIRST VENDORS
Sell the shiny chatbot on top of unready data, leading to confident hallucinations.

At Celgene, **seven of our engineers supported** a \$10 billion-a-year commercial portfolio. That was pre-AI. With Claude Code in our stack now, we can do it twice as fast with half the staff.

TestGen generates 120+ AI-driven data quality tests against any pharma dataset. DataOps Observability watches every pipeline. Both Apache 2.0. Both yours without us. We use **FITT data architecture**, so the platform doesn't break at 2 am the night before a board meeting.

Typical staffing is one to three of our engineers. Flat monthly rate, set at scoping.

AI-ready data on your side

Your CEO wants a Snowflake Cortex or Databricks Genie chatbot included in the launch-readiness deck. Fine. Here's what nobody is telling them.

When MIT's NLP group tested leading text-to-SQL models against the Spider 2.0 benchmark of real enterprise databases, accuracy dropped from over 85% on academic benchmarks to under 20% on real warehouses. The model wasn't broken. The problem was that **the data underneath it had no context**. Commercial pharma data is the worst version of this. Your warehouse has 150 tables. Half are deprecated. The right join between specialty pharmacy and claims depends on a payer hierarchy that lives in someone's head. EQ_TRX, NRx, formulary, ALPD, SOB, 852, 867. The AI doesn't know any of it. It guesses, and it's confidently wrong. Your CEO sees it, loses faith, and your budget gets cut next quarter.

We engineer the context layer your AI tools need. Schema and grain. Business definitions for equalized prescriptions, territory alignment, and payer hierarchy. Validated example queries. Live freshness state on every IQVIA or ICON/Symphony refresh.

We're building this layer for a major pharma on Snowflake Cortex and Databricks Genie right now, covering specialty pharmacy and claims data for a recent launch. We work with the data you have: IQVIA, ICON/Symphony, Veeva, specialty pharmacy, medical claims, NPP, RWE, copay, EDI 852/867. No learning curve. Our **Commercial Data and Analytics Platform for Pharma** is built for this.

The framework is **DT + DX + CTX = 10x**. Data Trust + Data Experience + Context Engineering. Published. Open.

Your analysts get their week back

Picture your senior analyst on Monday morning. The brand team wants a revised forecast by Wednesday. Before she can touch the model, she reconciles last week's IQVIA refresh against claims, chases down why a payer name came through three different ways, and rebuilds a territory roll-up because someone changed the alignment file on Friday. By Wednesday, she's done the cleanup. The forecast slips to Friday.

GIVE YOUR ANALYSTS THEIR WEEK BACK

THE CONSULTING TRAP

M	T	W	Th	F
Mon-Wed: Data Janitorial Work. Reconciling IQVIA vs claims, fixing payer names, rebuilding territory roll-ups.			Thu-Fri: Rushed forecasting.	

Takeaway: 30-50% of an analyst's ZS/Trinity bill rate is wasted cleaning the warehouse.

THE ENGINEERED SYSTEM

M	T	W	Th	F
Monday: Instant data trust. Tested, observed, shaped.	Tue-Fri: Dedicated entirely to brand strategy, advanced modeling, and forecasting.			

Takeaway: Well engineered systems, with AI can cover **hundreds of data sets with perfect quality** with a tiny team.

That's 30 to 50 percent of your analyst bench dedicated to data janitorial work. At your fully loaded ZS or Trinity bill rate, that's the line item that should fund the brand instead of cleaning the warehouse.

When the data below is tested, observed, and shaped to answer the questions analysts are asked, the time comes back. At Celgene, 10 to 12 analysts covered hundreds of datasets without missed SLAs because the engineering layer held up.

Engineer your way to cost savings and a scalable future

At one pharma company, we replaced what they had and **cut their annual data costs by 58%**. Same people getting the answers they needed. Better answers, because trust went up. If you have a product already in market and you're trying to free cash for the next launch, **that takeover is its own thing**.

Flat monthly rate, set at scoping. We work in your cloud. The code is in your repo.

THE ARCHITECTURE OF FAST, AI-READY DATA

Top Layer: The Output

Snowflake Cortex / Databricks Genie

The CEO's AI chatbot finally works because the context is structurally engineered below it.



$$\text{DT} + \text{DX} + \text{CTX} = 10x$$

Data Trust, Data Experience, Context Engineering

We engineer schema and grain, define territory alignments, validate queries, and track live freshness state.

Well engineered data platform is DataOps + DT + FITT = 10x

Bottom Layer: Raw Inputs

IQVIA, Veeva, Specialty Pharmacy, Claims, RWE, Copay, EDI 852/867, ICON/Symphony.



The CFO comparison: six commercial data FTEs fully loaded in Boston or the Bay Area run \$1.5M to \$2M a year, plus a six to nine-month recruiting cycle you don't have. A multi-year analytics consulting agreement runs higher and never transfers anything back to you. We come in under both numbers. The platform belongs to you when we leave.

If your launch slips, the rate scales down with you. Thirty days' notice. No penalty for an FDA delay you didn't choose.

Trust, speed, value, and yours

Three of our customers were acquired by top-five pharma for \$100 billion combined. BMS bought Karuna for \$14 billion. BMS bought Celgene for \$74 billion. Merck bought Acceleron for \$11 billion. The platforms stayed with the customer every time. Eisai transferred the platform to Accenture cleanly when they were ready.

41 people from five companies have worked safely on the same data for seven years. Still in production. 253,485 lines of code. 594 automated tests. TestGen catches errors before they reach production.

Diagnostic: Renting Labor vs. Owning a System

	Status Quo	Engineered System
Business Model	Hourly Extraction (\$300/hr)	Flat-Rate System Build
Team Size	10-30 bodies	1-3 AI-powered engineers
AI Readiness	Blind / Hallucinating	Context Layer built-in
Risk Adaptability	Punished for FDA Delays	Scales down with 30-days notice
Endgame	Permanent Retainer / Lock-in	100% Transfer to your team or acquirer

Field reporting ready six months before launch, every time. First star schema in four weeks. First AI context layer in six. Full launch report set six months before go-live. The [launch calendar and the reports it has to feed](#) lays out every milestone, and the [drug launch case study](#) walks through one end-to-end.

The platform sits in your cloud. The code is in your repo. We transfer when you want. To your team. To Accenture. To your offshore center. No lock-in. No exit fees. No drama.

Every dollar you don't spend on data consulting is runway

We're a data engineering services company backed by an open source product. We send pharma data engineers, not generic consultants. We bring our own software and we work in your cloud.

If you want to read first, start with the [\\$100 billion acquisition story](#) or the [drug launch case study](#). They cover the work in more detail than this post can.

The Economics: Cut Data Costs by Two-Thirds

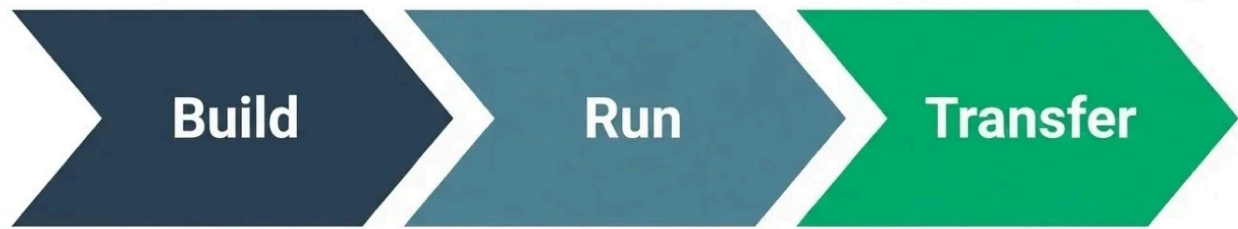


If your launch slips due to the FDA, the rate scales down with you (30 days notice). No penalty for delays you didn't choose.

If you **want to talk**, we can have two meetings to see if we're a good fit. First meeting, 30 minutes. You describe the launch, the team, the data sources, the timeline, and the AI pressure from above. We tell you whether we're a fit. If we're not, we say so and recommend who is. Second meeting, 60 minutes. We walk through one of your data sources end to end on your screen. We show what testing, observability, and context engineering look like on it. We show what your AI tool would answer if it were pointed at the right shape of data.

After that, if you want, there's a paid two-week scoping engagement. You walk away with a working context layer over one of your data sources, an AI-readiness scorecard, and a sized proposal. If you don't move forward, the work is yours.

The Asset is Yours: Zero Lock-In, 100% Transferable



The Philosophy

We bring our own software.
The code is in your repo.

The Proof (\$100B Track Record)

- BMS bought Karuna for **\$14B**.
- BMS bought Celgene for **\$74B**.
- Merck bought Acceleron for **\$11B**.

The Outcome

The platforms stayed with the customer every time. When Eisai was ready, they transferred the platform cleanly to Accenture.

No exit fees. No drama.

Every \$700,000 you don't spend on data consulting is runway. The company that survives the next two years is the one that wasn't running a fat back-office to begin with.

If your launch is 12 to 24 months out and you want to know what your data and AI bill could look like at one-third the run rate, [book the 30-minute call](#). The math is the conversation.

SECTION 3

Who should run your commercial data

1. The \$100 Billion Secret: Why Leading Pharma Companies Bring In a Dedicated Commercial Data Team
2. The Age Of Outsourcing Your Commercial Pharma Data Team To India Is Over
3. You Built the Thing. Now You're Stuck Running It.
4. We Have a Cadillac, We Need a Corolla: Cutting a Biotech's Data Bill to Fund the Next Launch
5. Cut the Data Run-Rate on the Product You Already Sell
6. Why I Chose DataKitchen for DataOps

The \$100 Billion Secret: Why Leading Pharma Companies Bring In a Dedicated Commercial Data Team

Read the original article at: <https://datakitchen.io/blog/the-100-billion-secret/>



Commercial pharmaceuticals, where billion-dollar decisions hinge on data-driven insights, success isn't just about having the right drug, it's about having the **right** data, delivered the **right** way, at the **right** time. Yet, too many pharmaceutical companies find themselves trapped in a frustrating cycle: brilliant analysts wait weeks for data extracts, IT teams are overwhelmed by competing priorities, and commercial teams make critical decisions based on incomplete or outdated information.

The \$100 Billion Secret: Why Pharma Leaders Outsource Their Data Teams

Common data challenges create bottlenecks for commercial pharma, but specialized, outsourced data teams deliver a significant competitive advantage.

The Problem: The Commercial Pharma Data Bottleneck

Overwhelmed IT Teams Create Critical Delays

Urgent data requests are queued behind competing priorities from across the company.



Analysts Receive Slow, Unreliable Data

Information is often incomplete or outdated, requiring extensive cleaning before analysis can begin.



Generalist IT Lacks Niche Pharma Expertise

They don't understand the nuances of specialty pharmacy, medical claims, or IQVIA data.



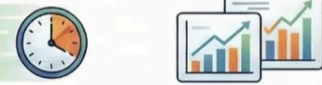
The Solution: DataKitchen's Specialized Approach

Proven Results: Supported \$100 Billion in Pharma Acquisitions

Our data infrastructure is trusted for the most significant commercial decisions.



Speed and Simplicity Provide a Competitive Edge



Get analyst-ready data in hours or days, not weeks or months.

Powered by a Platform for 10x Productivity

Our SaaS DataOps platform prevents errors, automates tasks, and ensures security.



The solution isn't more complex technology or bigger IT budgets. It's something far more fundamental: simple, integrated data managed by a specialized data team you can trust to deliver what you need, when you need it.

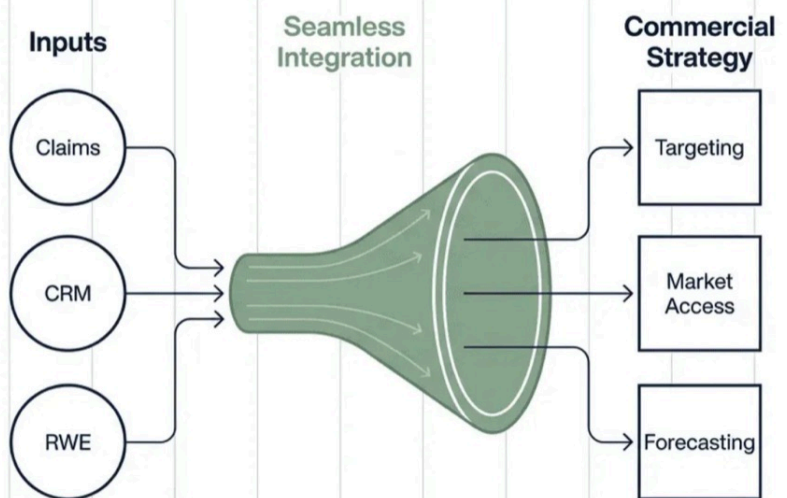
Understanding the Commercial Pharma Data Challenge

Before we explore the solution, let's examine why traditional approaches to pharmaceutical data management consistently fall short. Think of your current data landscape like a busy hospital emergency room. Everyone has urgent needs, but there's only one overworked IT team trying to serve every department, from clinical research to regulatory affairs to commercial analytics. Your request for integrated claims data gets queued behind the finance team's database migration and the regulatory team's compliance reporting system.

Billion-Dollar Decisions Hinge on “Analyst-Ready” Data

Success in today’s market isn’t just about having the right drug. It is about having the right data, delivered the right way, at the right time.

Too many **commercial teams are trapped in a cycle** where brilliant analysts wait weeks for extracts. Critical decisions on **Launch, Forecasting, and Sales** are made based on **incomplete or outdated** information.



This creates a cascade of problems that every commercial analyst is intimately familiar with. You spend more time chasing data than analyzing it. When you finally receive data, it’s often incomplete, inconsistent, or requires extensive cleaning before any meaningful analysis can begin. Meanwhile, your competitors are making faster market decisions because they’ve solved this fundamental data delivery problem.

The root issue isn’t technical capability, it’s focus and specialization. General IT teams, regardless of skill level, often lack the deep domain expertise needed to fully understand the nuances of specialty pharmacy data, medical claims integration, or real-world evidence datasets. They’re solving dozens of different problems across multiple business functions, while commercial analytics requires dedicated attention to a specific set of complex data challenges.

The Executive Summary: Solving the Commercial Data Bottleneck

The Challenge



Commercial teams are starved for insights because they rely on generalist IT teams. These internal teams operate like an “Emergency Room,” where commercial requests are triaged behind Finance, Regulatory, and HR.

The Solution



“Rent” a specialized DataKitchen team powered by a best-in-class DataOps platform. We focus exclusively on Commercial Pharma—we stay in our lane to make data simple, integrated, and trusted.

The Impact



- **Speed**: Move from months of scoping to days of delivery.
- **Validation**: Supported \$100 Billion in acquisitions (including Celgene and Karuna).
- **Control**: No black box. You retain full visibility and ownership of your data on your chosen cloud.

The Power of Specialized Focus

DataKitchen’s approach to commercial pharmaceutical data represents a fundamental shift in thinking. Rather than trying to be everything to everyone, they’ve chosen to stay in their lane, focusing exclusively on making commercial data easy and trusted. This isn’t a limitation, it’s their greatest strength.

Consider how this specialized focus translates into practical benefits for your daily work. When you need to integrate IQVIA market data with your internal sales performance metrics, you’re working with a team that understands exactly how these datasets should connect, what common integration challenges to expect, and how to deliver analyst-ready data that requires minimal additional processing. This isn’t generic data engineering, it’s commercial pharma data engineering, built on decades of specific industry experience.

Generalist IT Lacks the ‘Commercial Fluency’ You Need

The Generalist IT Team	The DataKitchen Specialist
 • Sees “files and databases” • Solves dozens of different problems across business functions • Lacks domain expertise, requiring you to explain basic industry concepts	 • Understands the nuances from Day One • Fluency in: Specialty Pharmacy, Medical Claims, RWE, and Payer Hierarchies • Result: No learning curve. We understand how IQVIA market data connects with internal sales metrics immediately.

This specialization extends deep into the technical architecture as well. The team understands that commercial pharma data has unique characteristics: it operates in environments that require strict governance, involves complex hierarchical relationships between products, payers, and organizations, is subject to time-sensitive market dynamics that demand rapid data refresh cycles, and necessitates integration requirements spanning everything from specialty pharmacies to genomics data. A generalist IT team might eventually learn these nuances, but a specialized commercial pharma data team brings this knowledge from day one.

Speed and Simplicity as Competitive Advantages

In commercial pharmaceuticals, speed isn’t just convenient, it’s often the difference between capturing market opportunity and watching competitors move first. DataKitchen’s emphasis on fast, simple, and inexpensive data delivery addresses one of the most persistent pain points in pharmaceutical analytics: the time lag between identifying a business question and having the necessary data to answer it.

Think about your typical data request process today. You identify a critical business question, perhaps analyzing the impact of a new competitor launch on your market share in specific therapeutic segments. You submit a request to IT, wait for scoping and prioritization, then wait again for development and testing. By the time you receive the data, the competitive landscape may have undergone significant changes. Months go by.

From Months to Days: Speed as a Competitive Differentiator

In a launch or competitive response, speed is value.

Traditional IT Request Cycle



DataKitchen Cycle



Speed compounds over time, enabling iterative analysis and faster hypothesis testing.

The DataKitchen model completely reverses this dynamic. Because they maintain integrated, analyst-ready commercial data as their core focus, responding to new analytical needs becomes a matter of hours or days rather than weeks or months. This speed advantage compounds over time, enabling more iterative analysis, faster hypothesis testing, and ultimately more responsive commercial strategies.

The simplicity component is equally important but often underestimated. Complex data architectures might seem impressive, but they often create maintenance overhead that slows future enhancements and increases the risk of errors. DataKitchen's approach prioritizes understandable, straightforward data representation that analysts can work with confidently, reducing the time spent interpreting data relationships and increasing the time available for actual analysis. You can pick the self-service insight tool that you love best.

Building Trust Through Transparency and Results

Trust in data teams doesn't emerge from promises, it builds through consistent delivery and transparent operations. DataKitchen's track record speaks directly to this principle. When they mention \$100 billion in customer pharmaceutical acquisitions, including significant transactions like Celgene and Karuna, they're not just citing impressive numbers. They're demonstrating that their data infrastructure and team have supported some of the most significant commercial decisions in recent pharmaceutical history.

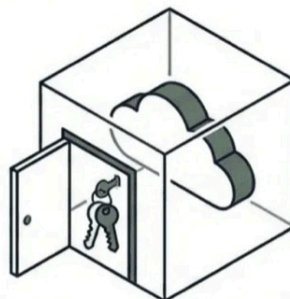
This track record matters because pharmaceutical acquisitions represent the ultimate test of data reliability. Due diligence processes scrutinize every aspect of a company's commercial performance, from market positioning to sales forecasting and competitive dynamics. The data infrastructure supporting these analyses must be not only accurate but also auditable, defensible, and comprehensive. Success in this demanding environment provides strong evidence of operational excellence.

A Partner, Not a Hostage Taker

No Black Box. Complete Transparency.

We work in your IT team's chosen cloud provider. If you decide to take over, we hand over the keys with no extra charges.

You aren't just renting a service; you are partnering with a team that builds transparent solutions.



Safety Record:

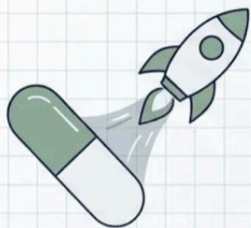
Validated by a customer where 42 people from 5 different companies worked safely on the data over three years.

The transparency aspect extends to DataKitchen's explicit commitment to avoiding black box solutions. When they state that the eventual IT takeover involves "no black box, no extra charges," they answer the usual worry about bringing in an outside team: that you never get the work back. The solutions are transparent and transferable by design, so your organization keeps full visibility into how its data infrastructure runs and stays free of vendor lock-in. One customer has had 42 people from 5 companies work safely on data over three years

Comprehensive Data Expertise Across Commercial Domains

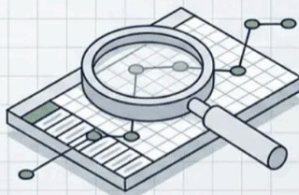
The breadth of DataKitchen's data experience reveals the depth of specialization required for effective commercial pharma analytics. Their expertise spans specialty pharmacies, medical claims, net price positioning (NPP), real-world evidence (RWE), Veeva CRM data, IQVIA market intelligence, and numerous other critical data sources. This isn't just a list of technologies, it represents years of understanding how these diverse data sources integrate to support commercial decision-making.

Proven Success Across the Lifecycle



Drug Launch

Hitting the ground running with clean, aligned data for new product planning.



Real-World Evidence

Integrating complex datasets to prove value and patient outcomes.



Field & Promotion

Optimizing Non-Personal Promotion (NPP) channels and Field Force dashboards.

Consider the complexity involved in creating a comprehensive view of commercial performance. Specialty pharmacy data provides insights into patient access and adherence patterns. Medical claims data reveals actual utilization patterns and outcomes. Data from sources like longitudinal patient data informs access strategy decisions. Veeva data increasingly influences targeting and positioning strategies. Each of these data sources has unique structures, update frequencies, quality considerations, and integration challenges.

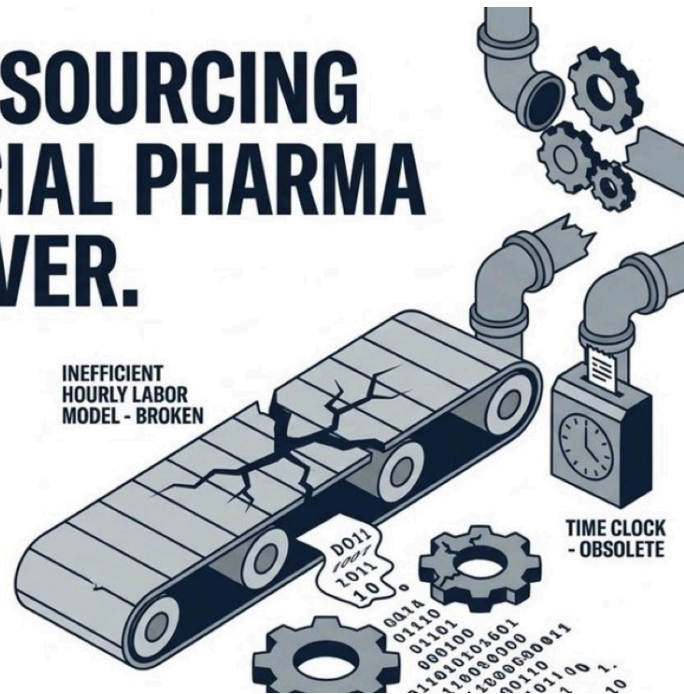
A general IT team might successfully integrate one or two of these sources, but understanding how they all connect to create actionable commercial insights requires specific pharmaceutical industry experience. DataKitchen's team brings this integrated perspective, understanding not just how to extract and load each dataset, but how to structure and relate them for optimal analytical utility.

The Age Of Outsourcing Your Commercial Pharma Data Team To India Is Over

Read the original article at: <https://datakitchen.io/blog/age-of-outsourcing-commercial-pharma-data-team-to-india-is-over/>

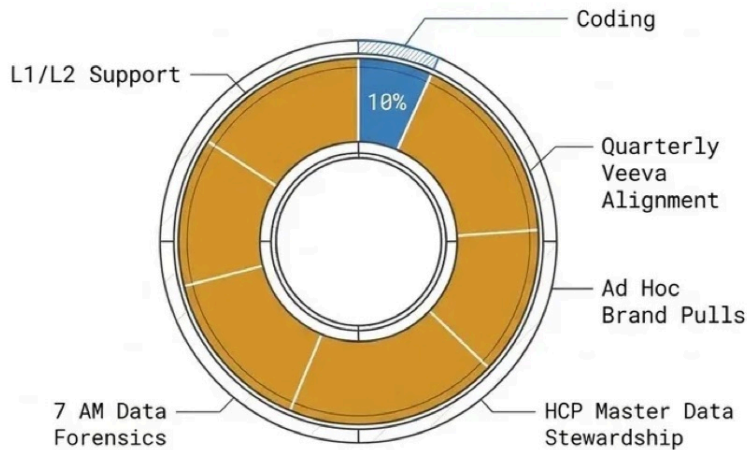
THE AGE OF OUTSOURCING YOUR COMMERCIAL PHARMA DATA TEAM IS OVER.

The offshore hourly model is spent. The new lever is AI agents in a better factory.



You run commercial analytics at a pharma company. Somewhere in your budget is a line for 20, 40, maybe 80 offshore developers. They write the ETL. They fix the ETL when the **Veeva feed changes**. They work a ticket queue that never gets shorter. You pay by the hour, and the hours never go down.

Offshore models discounted manual steps but never removed a single one.



“Every handoff, every manual deploy, every 2 AM re-run **still** happens. It just happens in a cheaper time zone. Same defects, lower labor rate.”

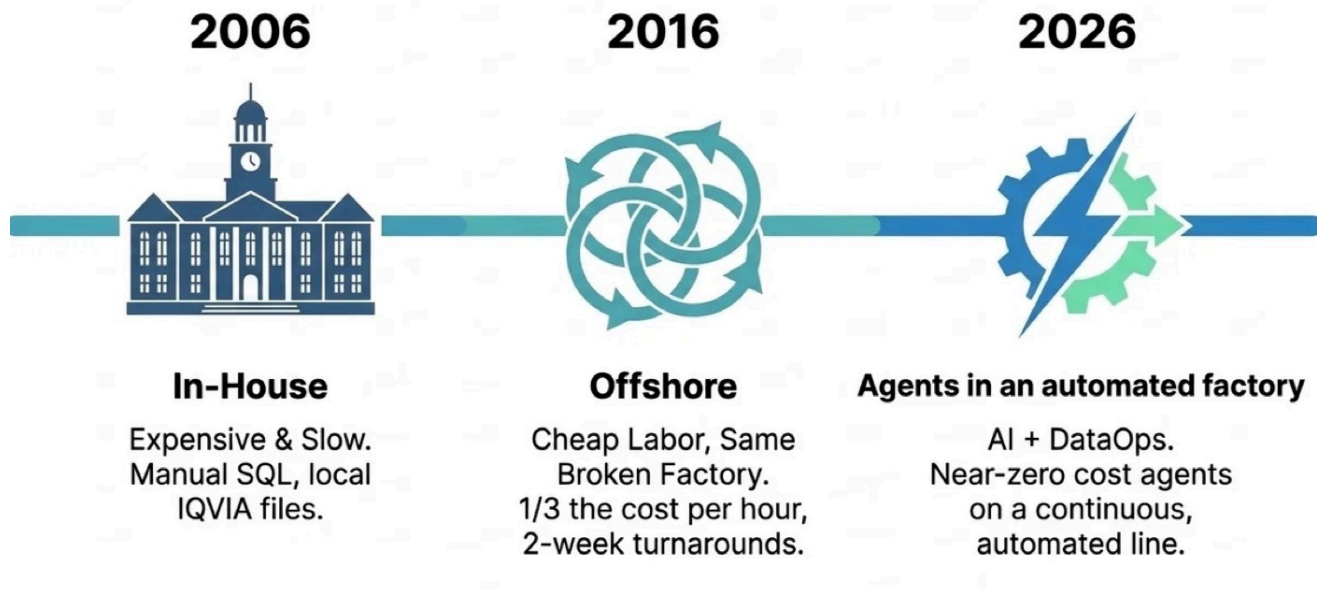
For ten years, the only lever you had on that line was the hourly rate. Cheaper workers on the same assembly line. That lever is spent.

Commercial pharma data engineering has had three ages: insource, outsource, and now agents. The first two argued about who wrote the code and what an hour of them cost. The third one changes the math. When the worker costs almost nothing, the only lever left is the factory the worker stands on. AI lets you do more. [DataOps automation](#) makes it cost less. Together they cut commercial data engineering labor by 70% or more, and the work comes out with fewer errors than the offshore team ever produced.

Agents in an automated factory

The first age was insource. In 2006, you built in-house. A team in Princeton or Cambridge wrote SQL by hand, loaded the IQVIA files on a schedule, and hoped the numbers matched what the field saw. Expensive and slow, but yours.

Three Eras of Commercial Data Engineering



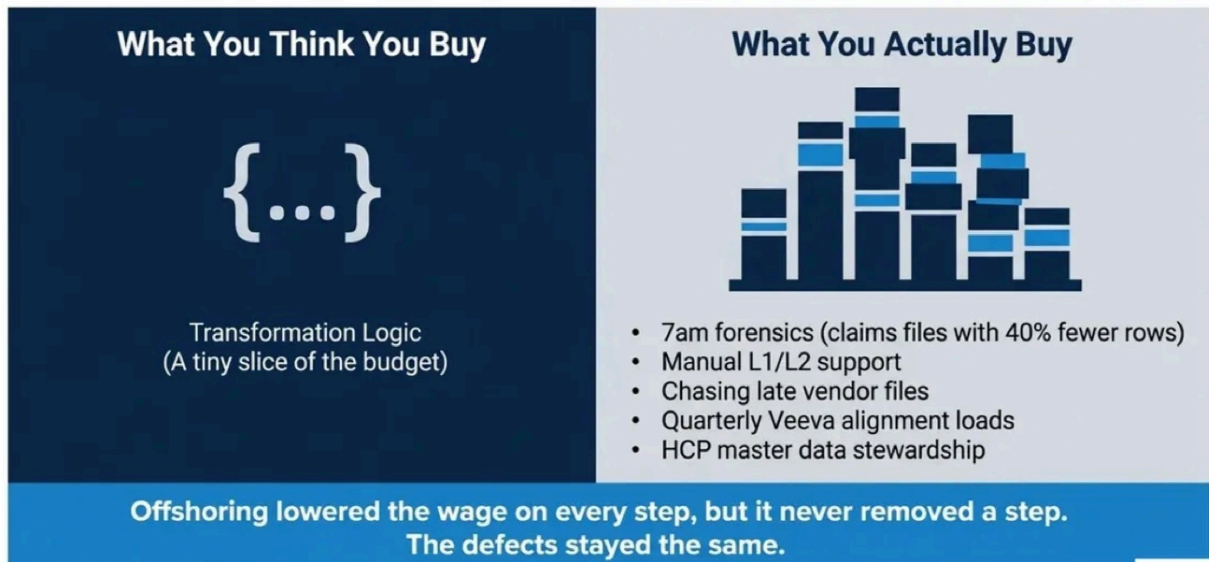
The second age was outsource. In 2016 you sent that work offshore. The math was simple. A developer in Hyderabad cost a third of a developer in New Jersey, so you bought hours in bulk. You also bought time zones, handoffs, requirements documents nobody read, and a two-week turnaround on a one-line change. The factory didn't change. You just paid less per hour to run it.

The third age is agents, and it is the one that changes the math. In 2026, the focus finally lands on the factory. **AI agents** took the cost of the workers, the people writing the code, to near zero. But a worker who costs almost nothing is only as productive as the line they stand on. Put an agent on the 2016 assembly line, and you get cheap code stuck behind the same manual deploys, the same ticket queue, the same 7am forensics, and a human reviewer who is now the slowest station on the floor. Put that agent on an automated line, where DataOps runs the code, tests it at every step, and catches it when it breaks, and the two multiply. Near-zero-cost workers on a line that never stops for a handoff. That is where the productivity comes from, and it's a very different headcount. The first two ages moved the price of an hour. This one removes the hours.

Cheaper workers vs. a better factory

Think about where the money actually goes. Coding is a small slice. The bulk is the manual work around the code. Gathering requirements. Translating them into a spec. Waiting for review. Pushing to production by hand. Finding out at 7am that the weekly claims file had 40% fewer rows. Tracking down who changed what, fixing it, re-running it, and explaining it to the brand lead.

Cheaper Workers vs. A Better Factory

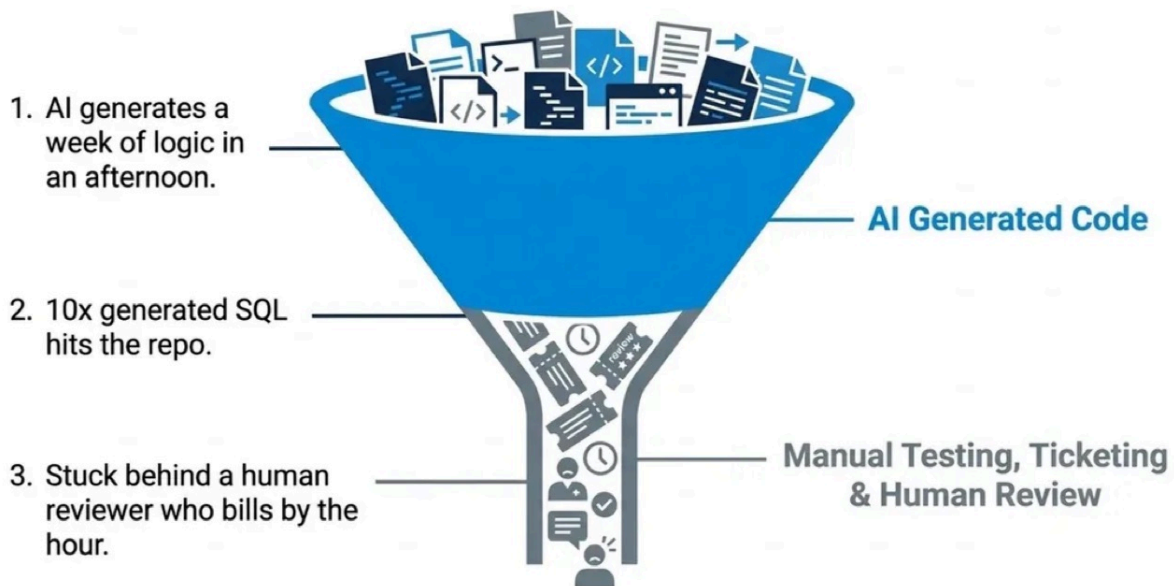


Look at what your offshore hours actually buy, and half of it isn't code at all. L1 and L2 support. Chasing the vendor about a late file. The quarterly Veeva alignment load. Ad hoc pulls for the brand team the night before a business review. [HCP master data stewardship](#). Offshoring attacked all of that by lowering wages at every step. It never removed a step. Every handoff, every manual deploy, every 2am re-run still happened. It just happened in a cheaper time zone. That's why quality never improved. Same assembly line, same defects, lower labor rate. The savings were an illusion. You paid less per hour to do the same number of hours.

AI takes the coding cost to near zero

A data engineer with a good coding agent can produce a week's worth of transformation logic in an afternoon. You already know this.

The AI Trap: Faster Bad Engineering



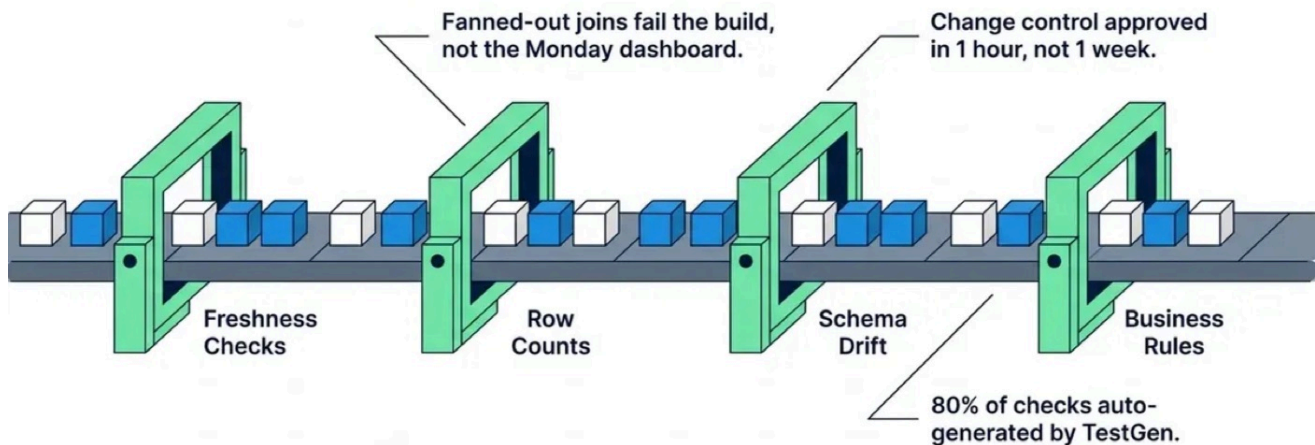
Here's the trap. AI alone speeds up one station on the line, leaving the rest untouched. Your offshore partner will happily sell you "AI-enabled" developers who still bill by the hour and still file tickets. The output goes up. The overhead doesn't move.

We put it this way in [Where to Actually Start with DataOps in 2026](#): AI didn't change what good data engineering means. It changed how fast bad data engineering happens. Four quarters after the agents show up, a typical repo holds about 10 times as much generated SQL as before, yet sits behind the same handful of tests. Everything in that gap was written without a check. Even teams with [idempotent loads](#) and clean review habits can't read code faster than an agent writes it. That's a faster line producing more defects, billed at whatever rate you negotiated.

Build an automated factory with AI agents as the workers

DataOps rebuilds the rest of the line so the faster station doesn't flood the ones behind it. Every pipeline runs with [tests on every table at every step](#). Freshness, row counts, schema drift, business rules, all checked automatically on every run. The difference from a monitoring tool is where the tests run: in CI, before the merge, against yesterday's data, so a fanned-out join fails the build instead of the Monday dashboard. Deployments are versioned and automated. When a vendor changes a feed, the test fails before the report does. Change control still exists, but it approves within an hour rather than a week because [the regression suite has already answered the question](#) the CAB would have asked. Nobody files a ticket. Nobody waits for the morning standup in Bangalore.

Build an Automated Factory with AI Agents



By removing steps instead of discounting them, labor drops by 70%.

Requirements change shape too. Instead of a 40-page spec that goes stale the day it's signed, you get acceptance tests that run forever. When sales ops says "a rep can't hold two territories in the same week," that becomes a test. When the IC analyst says "samples don't count toward quota," that becomes a test. **TestGen** profiles the data and generates about 80% of the checks. Your engineers write the 20% that encodes the business logic no profiler can infer, and that 20% is where the institutional knowledge that used to live in a contractor's head now lives in your repo.

So the coding cost approaches zero. The production overhead approaches zero. The requirements cost drops hard. Stewardship and the odd ad hoc pull still take a human, and that human should be one of yours. What's left is a small team of people who understand your brands, your data vendors, your field force, and your launch calendar. Somebody has to run the factory, own the test suite, and stop doing forensics, and that somebody is the manager who used to run the ticket queue. The savings come from removing steps, not discounting them. That's the 70%.

The Indian outsourcers already know their time is closing

Don't take our word for it. Watch the outsourcing industry. In June, **Harvard Business Review** argued that generative AI is eroding the labor arbitrage model that built the offshore business, with IT services the most exposed. **Quartz** reported that clients are reopening multi-year contracts within two years of signing because AI changed the economics mid-deal. **Tata Consultancy Services now runs about 80%** of its BPO contracts on outcome-based pricing, double the share since 2023. **Infosys shed more than 8,000 workers** in a single quarter, even as revenue grew.

The Giants Know The Hourly Model is Over



Generative AI is actively eroding the labor arbitrage model. Mid-deal contracts are being reopened.

80%

The share of TCS BPO contracts now moving to outcome-based pricing.

8,000+

Workers shed by Infosys in a single quarter while revenue continued to grow.

They figured out the hourly model is dead. Are you still paying for it?

The giants are booking bigger profits with fewer people. They figured out the hourly model was over. The question is whether you're still paying for it.

We have run commercial data teams this way for years, and the evidence is public

One small biotech **cut its data bill by two-thirds** when we replaced what they had. One and a half of our engineers built and ran the launch data platform for **Cobenfy**, and BMS bought the company for \$14 billion. **Three of our customers** have been acquired for nearly \$100 billion combined.

DataKitchen's Track Record of Transformation

66% Cut

Replaced legacy systems for a small biotech, cutting their data bill by two-thirds.

\$14 Billion

Just 1.5 DataKitchen engineers built the launch data platform for Cobenfy, leading to a \$14B BMS acquisition.

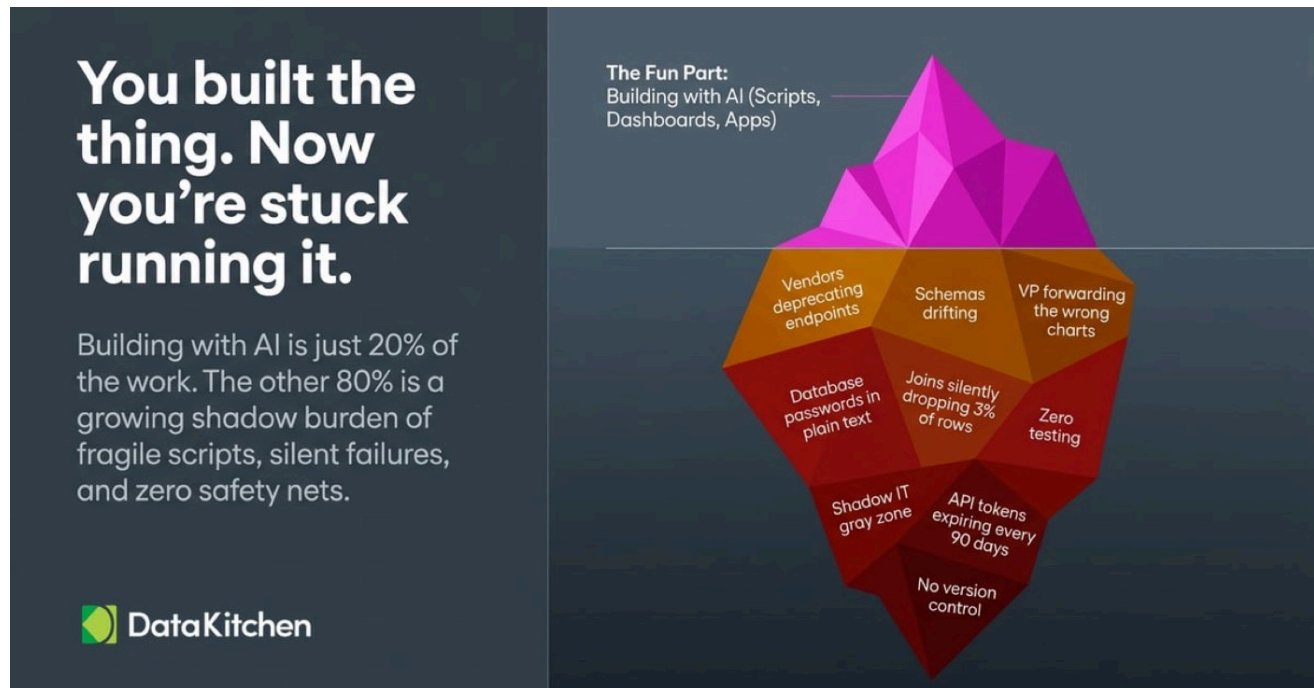
\$100 Billion

Three DataKitchen customers acquired for nearly \$100B combined.

If you're **pre-launch and watching cash**, this matters more. Every dollar going to offshore hours is a dollar not going to the launch.

You Built the Thing. Now You're Stuck Running It.

Read the original article at: <https://datakitchen.io/blog/you-built-the-thing-now-youre-stuck-running-it/>

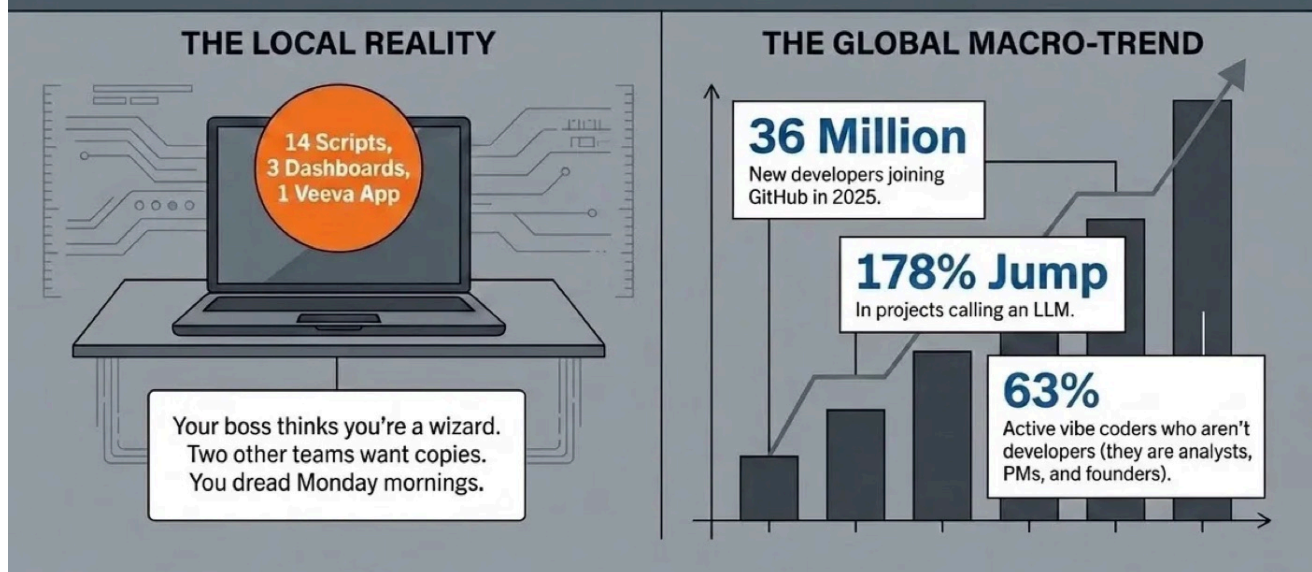


Six months ago you couldn't write a line of SQL. Now you have 14 scripts, three dashboards, and a small app that pulls call notes out of Veeva and scores them. You built it all yourself, with an AI coding tool and a lot of "no, the other date column." Your boss thinks you're a wizard. Two other teams have asked for copies.

And you have started to dread Monday mornings.

You are not alone in this. GitHub's [2025 Octoverse report](#) counted more than 36 million new developers joining in a single year, over 230 new repositories created every minute, and a 178% jump in projects that call an LLM. A lot of those new developers are people like you. In one [analysis of the vibe coding community](#), about 63% of active members weren't developers at all. They were analysts, product managers, ops leads, and founders, and roughly [one in five things they built was an internal business tool, and another one in eleven was a dashboard](#).

Six months ago you couldn't write SQL. Now you're a one-person IT department.

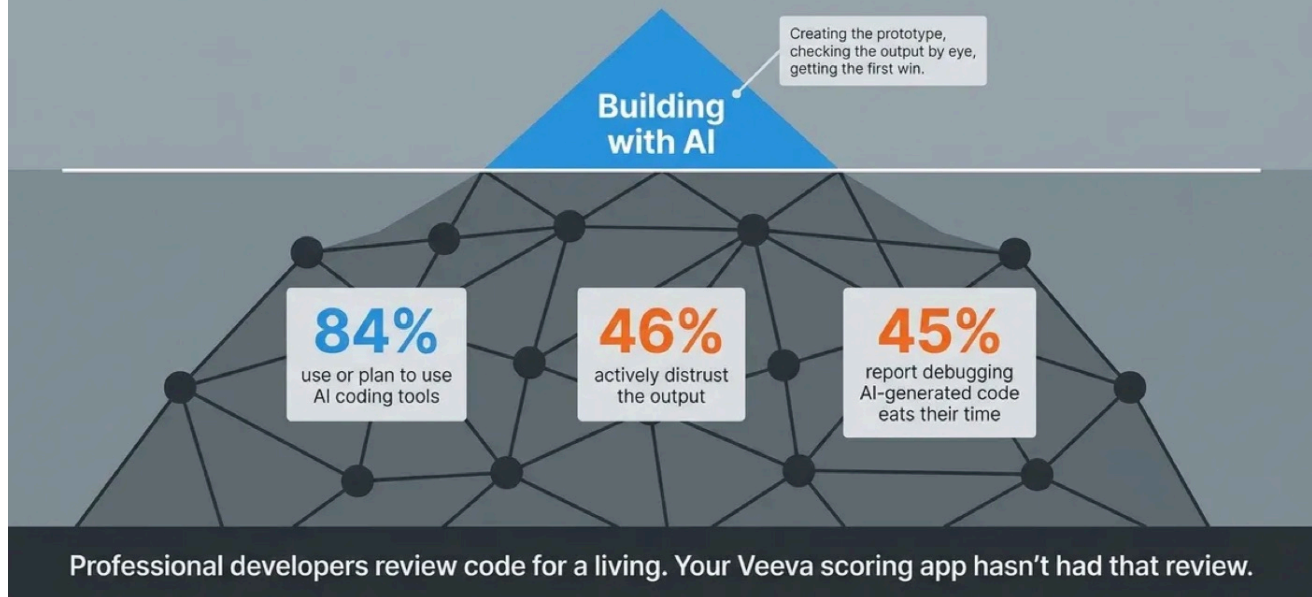


The quality data is less cheerful. In [Stack Overflow's 2025 Developer Survey](#) of 49,000 developers, 84% said they use or plan to use AI coding tools, but only 29% trust the output to be accurate and 46% actively distrust it. Their top frustration was code that is almost right but not quite, and 45% said [debugging AI-generated code](#) eats their time. Those are professionals who review code for a living. Your Veeva scoring app almost certainly hasn't had that review.

Building is 20% of the work. Running it never ends.

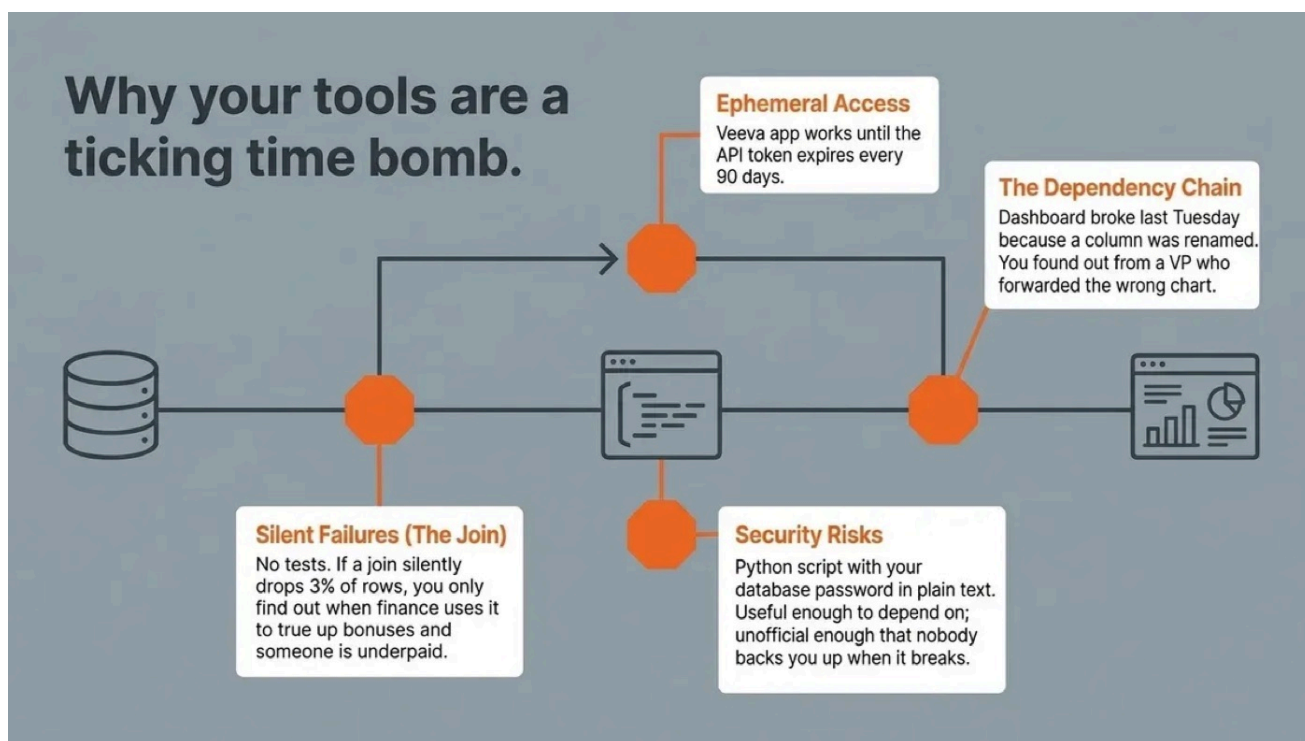
Here's what nobody told you about vibe coding. Building the thing is the fun part. It's also about 20% of the work. The other 80% is [keeping it alive](#), and that part never ends.

Building is 20% of the work. Running it never ends.



Think about what your week actually looks like now. The territory alignment script runs when you remember to run it, which means it runs when someone asks why the numbers are stale. The dashboard that pulls from the data warehouse broke last Tuesday because **someone renamed a column**, and you found out from a VP who had already forwarded the wrong chart to her boss. The Veeva app works, unless the API token expires, which it does every 90 days, which you learned the hard way. You have a Python script with your database password in plain text. You know that's bad. You haven't had time to fix it.

None of these tools **has a test**. Not one. When you built them, you checked the output by eye, saw a number that looked right, and moved on. That was fine for a prototype. It's not fine for a report that finance now uses to true up bonuses. If the join **silently drops 3% of rows**, you won't know until someone gets underpaid and complains.



Then there's IT. You haven't told them about most of this. Not because you're hiding it, but because the first conversation would be about why you're running production code off your laptop, and the second would be about a six-month project to "do it right," and by then the business would have moved on. So the tools live in a gray zone. Useful enough that people depend on them. Unofficial enough that nobody will back you up when one of them breaks.

And they will break. That's not a knock on your code. All code breaks. Schemas change, vendors deprecate endpoints, a source system gets migrated, a laptop gets replaced. The difference between a professional data team and you isn't that their code is better. It's that they have **machinery around the code** that catches the break before a VP does.

Here's the part that stings. The more useful your tools are, the worse this gets. Every new dashboard someone asks for is another thing you own forever. Every copy you hand to another team is another version that will drift, and none of it is in source control. When finance asks which version produced last quarter's number, you won't know. When a change breaks something, there's **nothing to roll back to**. You are becoming a one-person IT department for a portfolio of software you built on the side, and the job you were hired for is still there waiting.

You have three options, and none of them are good

Keep going

Wake up early, babysit the scripts, fix what breaks, hope nothing important fails during your vacation. This works until it doesn't, and when it doesn't, it fails in public.

Hand it all to IT

They will be polite about it. Their caution is earned, too: they're the ones who get paged when it breaks, who answer to audit when an unreviewed script **touches patient data**, and who inherit it when you move on. So they'll put it in a queue, scope a rebuild, and deliver something in 2027 that does less than what you have now. The speed that made these tools valuable in the first place is gone.

Hire an outsourcer

They'll **take it over by the hour**. You'll lose the ability to open the code and change it yourself, because now it's theirs. And their bill has no reason to ever get smaller, because smaller bills mean less revenue for them.

You have three options, and none of them are good.

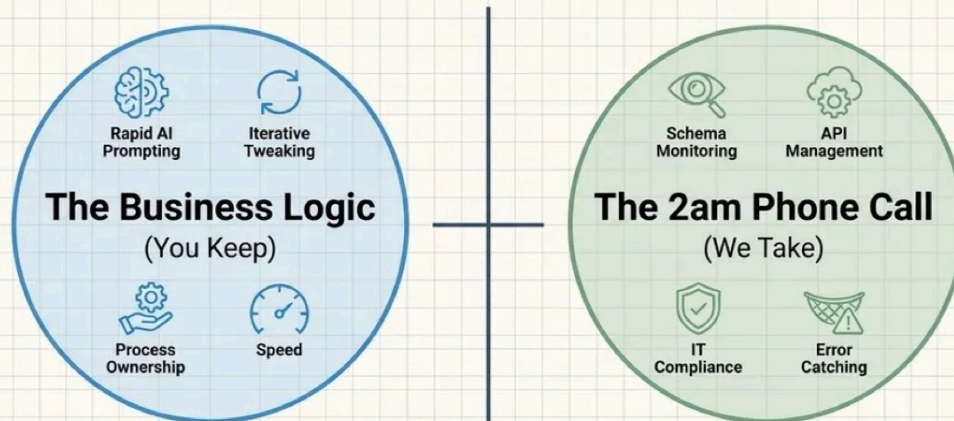
Keep Going (Status Quo)	Hand to IT	Hire Outsourcer	Vibe to Prod (DataKitchen)
			
Fails in public. High burnout, babysitting scripts, 2 AM panics on vacation.	2027 delivery. Scope a rebuild, sit in a queue, lose the speed that made the tool valuable.	Expensive lock-in. Lose access to your own code. Hourly billing incentivizes inefficiency.	Instant relief, retain ownership, enterprise reliability.

Keep the logic. Hand off the 2am phone call.

What you actually want is simple to describe and, until recently, impossible to buy. You want to keep building. You're good at it and it's the reason your team is ahead. You want someone else to run what you build, watch it, fix it when the schema changes, and answer to IT and security so you don't have to. And you want to keep the right to open the code six months from now, improve it with **whatever AI tool you like**, and hand it back without waiting in a queue.

Ownership of the ideas and the business logic stays with you. Ownership of the 2am phone call goes to somebody else.

The master key: decouple the logic from the liability.



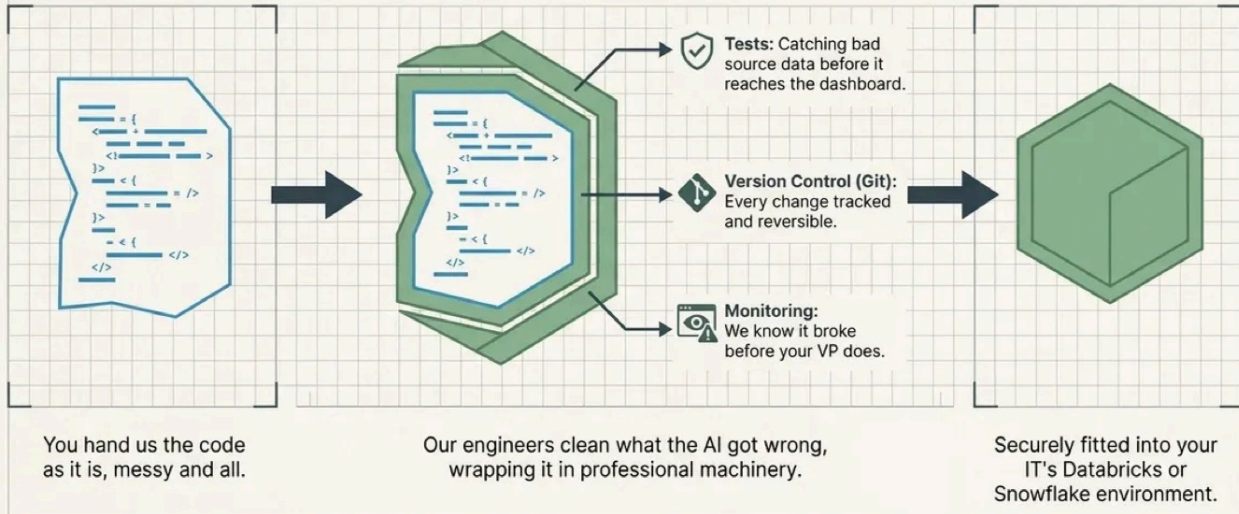
You want to keep building. You want someone else to watch it, fix it, and answer to security. And you want the right to open the code six months from now, tweak it, and hand it back without waiting in a queue.

Vibe to Prod: we make your messy code real

That's what DataKitchen does. We call it **Vibe to Prod**. You hand us the code as it is, messy and all. Our data engineers clean up what the AI got wrong, then wrap it in the machinery a professional team would have built from the start: **tests on the inputs and outputs**, git and version control so every change is tracked and reversible, **monitoring so we know it broke before you do**, and **checks that catch bad source data** before it reaches your dashboard. We fit it into your Databricks, Snowflake, or whatever your IT team runs, and operate it from then on.

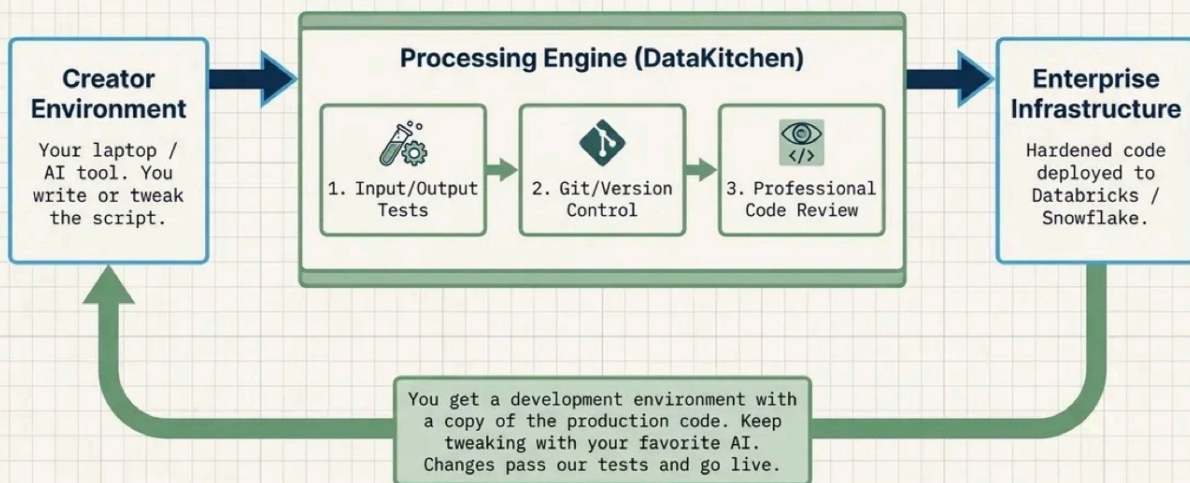
You get a development environment with a copy of the production code so you can keep tweaking with your favorite AI tool. Changes pass the tests, get a quick review from one of our engineers, and go live. We are experts on **efficient data operations**.

Introducing Vibe to Prod. We make your messy code real.



We've done this before. For one biotech we took over code that already existed, rebuilt it, and ran it for a fraction of what the previous vendor charged. The details are in [Cadillac or Corolla: cutting a biotech data bill](#).

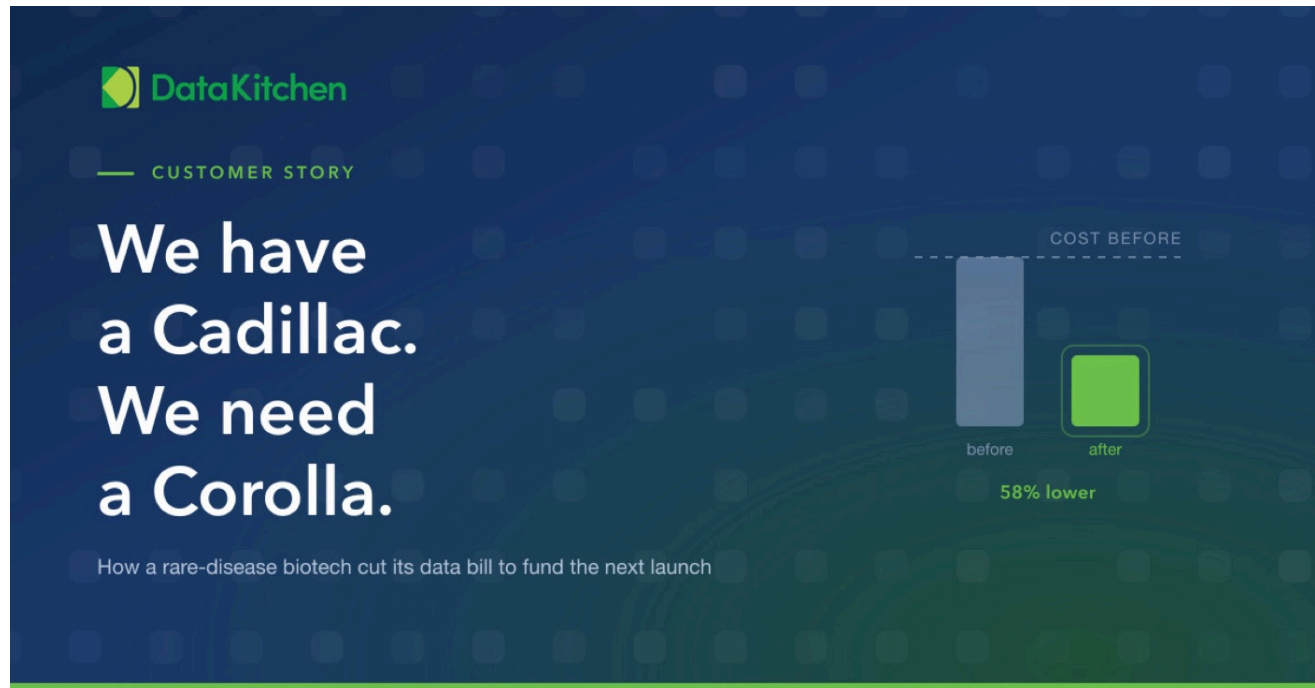
The machinery of stability.



You built it. We make it real. [Talk to us](#) if you have a pile of scripts that needs someone to mind it and manage it.

We Have a Cadillac, We Need a Corolla: Cutting a Biotech's Data Bill to Fund the Next Launch

Read the original article at: <https://datakitchen.io/blog/cadillac-corolla-cutting-a-biotech-data-bill/>



That was how the VP of technology at a rare-disease biotech described his commercial data operation the first time we spoke. He had fewer than a hundred patients on therapy, a second indication in phase III that would take him into the thousands, a finite runway, and a commercial team of about five people.

He was asking for a cheaper data warehouse, not a better one, because every dollar he stopped spending on running today's data was a dollar available to launch tomorrow's product.

He said Yugo, and we knew what he meant, but Yugo is not actually what he needed. A Yugo breaks. What he needed was a Corolla: unglamorous, right-sized, cheap to run, and boring in the specific way that means nobody gets a 2am phone call. That distinction matters, because a data platform that saves money by becoming unreliable hands the savings straight back the first time a launch report is wrong.

If the trade sounds familiar, we wrote the general version of it in [your data and analytics bill is the easiest line to cut](#). This is what it looked like in practice.

The cloud bill was never the problem

His Snowflake averaged a bit over \$700 a month. AWS ran a few thousand more.

Set that against what sat on top: two vendors, six to eight people between them, working out to three or four full-time equivalents, billing by the hour. One owned ingestion and the warehouse. The other owned the datamarts and the BI reports.

So the platform cost a few thousand a month and the labor around the platform cost roughly \$30,000 a month. If you go looking for savings in your cloud invoice, you are auditing the small number.

The warehouse itself was about two years old, and at most 40% of it could be reused for the new indication. He was paying Cadillac prices to maintain something he would largely rebuild anyway.

Two vendors, one seam, nobody in charge of it

The split was conventional, and the problem it created is structural rather than anybody's fault.

When a report came out wrong, the reporting vendor said the data arrived wrong. The ingestion vendor said the data matched the file spec they were given. Both were usually right. The only person who could see end to end was the director who had to referee it, and refereeing took him a couple of hours a day.

Figure 1: The conceptual architecture as we found it. Five layers, two owners, and a dashed line in the middle that neither contract covered.

We want to be fair about this, because it matters for anyone reading their own vendor scorecard: neither vendor was incompetent. One of them, by our customer's own assessment, took blame for problems that were not theirs. The specialty pharmacy sending four files a day was the single biggest source of trouble, and it was slow to correct anything. Getting one fix landed took several rounds, and nobody in the chain held a service level agreement worth invoking. That pattern is common enough that we wrote it up separately in [the commercial data most likely to be wrong](#).

What these files actually are

The specialty pharmacy sends four files a day: patient status, dispense, inventory, and demographics. The status file is an end-of-day snapshot, so a patient who moves through every stage in a single day arrives as four separate records.

That detail is the whole character of rare-disease commercial data. You are not sizing market share off a national prescription feed. You are tracking a few dozen people through consent, prior authorization, copay assistance, shipment, and refill. A wrong status on one of them is a person who does not get their drug on time. At this patient count, one bad record is a measurable fraction of the brand.

They influence prescribers without detailing

One assumption to correct, because it changes how every number in this post reads: this company has a commercial operation. It reaches prescribers through channels other than a rep knocking on doors.

For a few hundred patients scattered across thousands of possible prescribers, a traditional field force does not pay for itself. So the work runs through HCP websites, targeted display against NPI lists, third-party medical email, and medical liaison activity recorded in the CRM rather than sales calls.

That model is more data-dependent than detailing. A rep hands you a call note: one row, one prescriber, one date, already attributed. Non-personal promotion hands you thousands of anonymous digital events that somebody has to resolve to an NPI, reconcile against the HCP master, and then join to patient status to find out whether any of it moved a single patient. In that model the daily media analytics feed and the branded-versus-unbranded split in the website data are the measurement itself.

So the objection that this only worked because the operation was small has it backwards. The reporting was cheap to run because the work was automated, not because there was little of it. Attribution across a dozen digital channels with no rep to anchor it is the harder version of the problem, and it is the version more small brands are going to be running.

So they turned off the alarms

The warehouse shipped with 150 automated data quality checks. About 20 were still switched on when we arrived.

Nobody decided to stop caring. The checks fired every morning, most failures traced to the same handful of upstream problems, and every failure stopped a file. So the team did what any team does when the alarm goes off daily for a month. They disabled the loudest ones and kept the few they could live with.

By then one report was showing an email open rate of 181% and nobody had flagged it, which tells you how closely anyone was reading the reports. An analyst on the commercial team had quietly started pulling from the CRM to build her own numbers. She never circulated them. She just needed something she could defend in a meeting.

That is the part of bad data that never appears on an invoice. You pay two vendors to produce reports the business has stopped believing, while the person closest to the data rebuilds them by hand.

Cutting reports cut the bill, not the cost

He had already taken two rounds of savings before we got there, and both worked the same way.

The refresh moved from daily to weekly, and hours dropped. The report count went from 41 to 10, and hours dropped again. Of those 41 screens, his own estimate was that eight mattered, so the pruning was justified.

What never changed was the cost of producing one report. The savings ran out exactly when the reports did, and the next round would have to come out of the 10 that were left. That is the end of the road for cutting scope. Eventually you reach the reports somebody uses.

This matters for reading the cost number later in this post. We measure the 58% against the 10, not against the 41. By the customer's own count nobody was reading the other 31, so the comparison is a real workload against a real workload rather than a full operation against a gutted one. And the workload

has grown since: the same team absorbed the second indication's data work without the hours going up.

The line item nobody had questioned

Here is the one that generalizes furthest. A large share of the old bill existed because everything ran as regulated data.

That is the correct price for clinical data. It is the wrong price for sales reporting, specialty pharmacy status files, and web analytics. Validated environments charge change control, documentation, and qualification overhead on every change, and applying that to a weekly commercial dashboard means paying clinical-grade process cost for something no regulator will ever ask to see.

Separating the two, so each carries the controls it actually needs, was a real piece of the savings before anyone rewrote a line of code.

One vendor, fixed price, and a budget line for making it better

They consolidated to a single vendor and took the transition in 30 days of planning followed by an 11-week fixed-price implementation, which put the handover itself in 2026. We took over everything between the vendor SFTP drop and the BI report: ingestion code, the warehouse, transformations, datamarts, the reports themselves, the credentials, and the daily triage when a file lands late or wrong.

We did not win this on price. We came in a little higher than the incumbents' discounted renewal, and he chose us with his eyes open, because the discount was another round of scope cutting and he had run out of scope to cut.

After cutover, a standing team of about two people runs it. One on operations, one on automation and improvement. That second person is the whole argument. Without a budget line for making things better, a managed service is the same hourly triage with a different logo on the invoice.

The rate is flat and set at scoping, and it scales down if the launch slips, on 30 days notice. Nobody gets penalized for an FDA delay they did not choose. We work in their cloud and the code sits in their repo, so none of this is hostage to us staying.

What is not in there: no claims data, no IQVIA or Symphony, no formulary feed. For an in-market rare-disease brand with a handful of patients per prescriber, targeting off a claims universe matters far less than knowing exactly where each enrolled patient is today. That calculus changes with the second indication, which takes the population from under a hundred into the thousands, and we scoped the claims and physician-reference work for it rather than retrofitting it later.

How the number actually came down

Consolidating vendors removes the seam. On its own it does not make the work cheaper. What made it cheaper was changing how the work gets done.

Our engineers run Claude Code inside our **DataOps, FITT, and data testing framework**, so the AI writes the SQL transforms and the data quality tests while the engineers spend their time on what to build and on talking to the customer. **TestGen** generates tests against the datasets instead of waiting for somebody to hand-write them. **FITT data architecture** is why the pipeline does not fall over at 2am the night before a board meeting.

Figure 2: The same operation on a FITT architecture. Raw to final, one owner end to end, and the three middle layers gone along with the seam that ran through them.

That is also the real answer to the 150 rules nobody could live with. Generated tests are cheap enough to keep. A generated test that fires on a genuine upstream problem gets escalated to the vendor sending the file, rather than switched off because triaging it by hand costs more than the error does.

This is the move from buying analyst hours to owning an engineered system, and it is the whole reason the savings do not expire.

Be precise about the goal here, because “we reduced hours” is what every vendor says at renewal. Cutting hours by making people work faster is a treadmill. Those hours come back the week a new feed arrives or somebody asks for one more report, because the work was never removed, only compressed.

The goal is to delete the work. A file that gets patched by hand every month is a permanent human obligation, and you are paying for it forever whether the invoice says so or not. The same file with a generated test, an automated rule, and a fixed upstream cause is work that nobody has to do again, this month or in three years. Efficiency means the hour is gone, not that somebody is running faster through it. That is also why the savings survive the business asking for more: automation absorbs the next dataset, and a bigger bench does not.

What it produced

The annual cost of running that commercial data fell by about 58%.

Monthly hours have trended down since cutover. In the same period the team absorbed the data work for the second indication, so the reporting load went up while the run cost came down.

The issue log tells the same story from the other end. Under the prior vendors it recorded 30 data issues in 2024, from May when the log opens, and 43 in 2025, seven of which were still open on the day the operation transferred. Since the handover, it records none.

Zero needs explaining, because it is not a claim that the incoming files got clean. They did not. The specialty pharmacy still sends what it has always sent. What changed is where a bad file stops. The generated tests catch it in the pipeline, the fix goes in, and the data is published after that, not before. The log counts issues that reached somebody. Under the old arrangement a bad file became a wrong number, then a ticket, then several rounds with the vendor. Now it becomes a failed test and a change nobody downstream ever sees.

Figure 3: To be clear about the attribution, the 2024 and 2025 issues are not ours. They were logged under the prior vendors, before DataKitchen was involved at all. The operation transferred to us in 2026, and the log has stayed empty since.

At the most recent quarterly review, the customer's own summary was that he likes how quality fixes get operationalized instead of patched for the month. That distinction is why the number holds, and it is the same argument we make about [testing commercial pharma data](#), measured in dollars instead of defects.

The savings also bought something the old operation could never have carried. With the data engineered and tested underneath, we built them an agent over it so the commercial team can ask questions in plain language instead of queueing a report request. Point a chatbot at an untested warehouse and it [answers confidently and wrongly](#). Point it at this one and it has context to work from.

We also earned something that does not show up as a cost saving: permission to talk to the specialty pharmacy directly about its data quality. Nobody hands a new vendor that access in month one.

And the money went where it was supposed to go. The savings from running the approved product's data are funding the data work for the indication that takes this company from under a hundred patients to thousands.

If this looks like your operation

Forget the cloud bill. The tell is the ratio between hours your team spends proving whose fault a number is and hours spent making the number better.

If ingestion and reporting sit with different vendors and both bill hourly, that ratio works against you and gets worse as you add data. If your commercial analytics runs under clinical-grade controls, you are paying for assurance the work does not require. And if somebody on your team keeps a private spreadsheet they trust more than the dashboard, the reports have already lost.

Our [commercial pharma analytics practice](#) exists to bring in a dedicated commercial data team and take that layer over, which is the pattern behind [the \\$100 billion secret](#) and [how DataOps is transforming commercial pharma analytics](#). If you already have a product in market and you are trying to free cash for the next launch, [taking over an existing operation is its own offer](#). For the methodology rather than the pitch, [Data Quality: The DataOps Way](#) is the full version.

Cut the Data Run-Rate on the Product You Already Sell

WHO SHOULD RUN YOUR COMMERCIAL DATA

Cut the Data Run-Rate on the Product You Already Sell

The DataOps Way to Commercial Pharma Data

DataKitchen

You have a product in market paying the bills and another one coming. The job is to free cash for the launch without touching the brand, the field, or the pipeline.

The money is in the plumbing under the product that is already selling. It is usually the largest line nobody has looked at, and an existing commercial data operation can run for about a quarter of what it costs today. This chapter is about where that money is, why cutting it does not make the numbers shakier, and why the saving holds.

What does not get touched

This is not a commercial budget exercise. Everything that protects revenue or the pipeline stays exactly where it is: brand spend and the promotional budget, field headcount and territory coverage, the pipeline programs and launch prep, the syndicated data contracts, and the BI tool and dashboards people already use.

That last one matters. If cutting cost means retraining a field force on new screens, the cost came back in a different column.

The arithmetic

Look at where the money goes today and what happens to each line. Note which line does not move. The syndicated data contracts are the biggest number and the one everybody assumes is the target. They are not. The cost is in what it takes to run the data, not in buying it.

Line	What you pay for today	After
People	A bench of commercial data engineers, or an offshore pod, or both, plus whatever internal IT time gets charged against the brand.	One to three engineers. The structural change is that automation carries the load a headcount used to.
Consulting retainer	A multi-year analytics agreement billing analyst hours at around \$300, which never transfers anything back to you.	Gone. A flat monthly rate set at scoping, with the platform in your repository from the first sprint. The rate does not go up because the work took longer.
Software	Per-seat data quality and observability licenses layered on top of the warehouse you already pay for.	Apache 2.0 software underneath, running in your cloud. No per-seat license and nothing that expires when the engineers leave.
Syndicated data	IQVIA, Symphony, specialty pharmacy, claims. The largest line, and the one everybody assumes is the target.	Unchanged. You work with the feeds you already buy. This is not a data-contract renegotiation.
Analyst time	Thirty to fifty percent of your analyst bench doing data janitorial work at a fully loaded analyst rate.	That time comes back to analysis. It is the saving that never shows up as a line item and costs the most.

The People line is the one that changed shape. For ten years the only lever on an offshore pod was the hourly rate, and that lever is spent. The chapter on why the hourly offshore model is over covers what replaces it.

One of these has already happened

A small biotech with fewer than a hundred patients on therapy, a second indication in phase III, and a finite runway between the two. The data operation was the one line the head of commercial could move without touching the brand or the pipeline. His summary of what he had: "We have a Cadillac. We need a Corolla."

What he was paying for: two vendors, six to eight people between them, billing by the hour and working out to three or four full-time equivalents. One owned ingestion and the warehouse, the other owned the datamarts and the reports. Snowflake averaged a bit over \$700 a month and AWS a few thousand more, so the infrastructure was never the cost. The warehouse was about two years old, and at most 40 percent of it could be reused for the second indication. One report showed an email open rate of 181 percent that nobody had flagged, and an analyst on the commercial team was quietly rebuilding the numbers out of the CRM.

What replaced it: 30 days of planning, then an eleven-week fixed-price implementation covering everything between the vendor SFTP drop and the reports. A standing team of about two, one on operations and one on automation and improvement. The second person is the argument, because a

managed service with no budget line for making things better only ever gets more expensive.

The annual cost of running commercial data came down about 58 percent, with monthly hours still falling after the second indication's data work was added on top. That figure is measured against the ten reports the operation had already been cut back to, not the 41 it started with. By the customer's own count the other 31 were not being read, so this compares a real workload against a real workload rather than a full operation against a gutted one.

At the most recent quarterly review his own summary was that quality fixes get operationalized instead of patched for the month. That is why the saving holds instead of running out. And the answers got better while the cost came down, because reconciliation ran on every feed instead of the ones somebody remembered to guard.

Ask how the vendor gets paid

Ask how a vendor gets paid before you believe what they tell you about cost. An hourly agreement pays more for taking longer, staffing wider, and keeping what it learns. A flat rate pays the same either way, so the cheapest operation the vendor can build is the one they want to build.

That changes what gets automated. On a flat rate, every test an AI writes and every failure the platform triages by itself is the vendor's margin. On an hourly agreement those same hours are revenue somebody has to protect, which is why hourly vendors rarely automate them away.

Two more things follow from the same logic. The code should be in your repository from the first sprint, on Apache 2.0 software running in your cloud, so there is no version of the arrangement where the vendor knows something you do not and charges you for the gap. And the endgame should be handing the operation back, to your team, your offshore center, or another vendor, with no exit fee. A firm whose revenue depends on staying does not write that sentence down.

Cheaper usually means riskier. Here it does not.

The product in market is funding everything, including the launch you are saving for. If cutting its data cost made its numbers shakier, the trade would not be worth making. It does the opposite, and that is the whole reason this works.

At the pharma where we replaced the existing setup, costs fell by roughly three quarters and the answers got better, because reconciliation on every feed meant the numbers held when someone challenged them.

The in-market product stays the priority because it is paying for the pipeline. Seven of our engineers supported \$10 billion in US sales across Revlimid, Otezla, and Immunology at Celgene, with ten to 12 analysts covering hundreds of datasets and no missed SLAs. Only 18 percent of that engineering time went to data loading. The rest went to new analyst work, which is what a bench of loaders never gets to.

How a takeover runs

There is no cutover date. Feeds get taken over one at a time with tests on them first, so there is never a single day when everything has to work.

Assess. Measure what is running now: which feeds arrive, what they cost, what tests exist, and which reports the field actually depends on. The output is a coverage number and a finding list you did not have before, whether or not you go further.

Take over. Inherit the platform feed by feed, tests first, so coverage exists before anything changes underneath it. The incumbent can keep running while it happens. Nothing gets switched off on a date somebody picked in a plan.

Run leaner. One to three engineers operate what a bench plus a retainer used to. Generated tests, observability that triages its own failures, and a FITT architecture that does not need a person awake at 2am to survive a late vendor file carry the load. The result is a lower structural run-rate, not a one-time cost exercise.

Ready for the next one. The platform now running your in-market product is the one your next launch plugs into. The feeds are already tested, the dimensions already modeled, the team already knows your data.

Nothing you have built gets thrown away

Your warehouse stays where it is. The work happens inside your cloud account rather than moving your data somewhere a vendor controls. Your dashboards keep working, because what changes is what feeds them, not the BI layer your users have already learned. Your data contracts are untouched: same vendors, same files, same delivery schedule. Your incumbent vendor can run alongside the new team rather than being replaced on a cutover date, and can take the operation back later. And your scope can start with one product, or one feed, and widen once the run-rate moves.

A savings project ends. A lower run-rate does not.

A consultancy runs a cost-reduction project and leaves, and the run-rate drifts back within a year. Someone comments out the threshold that keeps firing. The hand-written test suite decays. The two people who understood the middle layer move on.

Changing the structure is what makes the saving persist: generated tests rather than a hand-written suite that decays, learned volume bounds rather than thresholds someone comments out, and a two-stage architecture with nothing in the middle to maintain. Nobody has to police the saving for it to hold, and nobody gets paid more for the operation needing more hands next year.

You do not pay twice

Most companies fund the in-market data operation and then fund a second capital project to build a launch platform beside it. The platform running your marketed product is the one the next launch plugs into: feeds already tested, dimensions already modeled, a team that already knows your data. Launch prep stops being a separate line.

That is the part nobody budgets for, and it is the largest number in this chapter.

Why I Chose DataKitchen for DataOps

Read the original article at: <https://datakitchen.io/blog/why-i-chose-datakitchen/>



Why choose DataKitchen? During my nearly seven-year tenure leading an analytics function at Celgene, our partnership with DataKitchen was a critical component of my team's data and analytics strategy. DataKitchen preaches the message of DataOps, a philosophy they shaped years ago to express their fundamental belief that focusing on data quality, automation, and accessibility at every stage in the data continuum will have a transformative effect on the organization. Let me just say with respect to DataOps - I'm a BELIEVER!

The "Why Choose DataKitchen" question for me always had two parts:

1. The belief that DataOps delivers on the transformation promise (it does!).
2. DataKitchen's ability to deliver DataOps benefits (they can!).

The impact of partnering with DataKitchen and establishing effective DataOps practices at Celgene was that the team did a whole lot more analytics, faster and with fewer errors than the other companies we benchmarked. This allowed the team to focus more energy on bringing value to our partners.

Organizations, such as Gartner, report that analytics and data science teams spend significant time and energy on transforming and correcting data, leaving much less time to do value-added analytics. That was my personal experience as an analyst, and I never met a colleague who didn't encounter those same challenges. Using DataOps, our team spent dramatically less time with data prep, rework, and error tracing, freeing the team to innovate, iterate, and improve.

There are many tools in the DataOps kit. From my perspective, testing, automation, and Agile practices laid the foundation for the team's success. Testing in the way DataKitchen tests completely changed our team's relationship with the data. In the DataOps world, testing is easy to set up and deployed across the data continuum. Data was tested at the point of origin and then at each stage, as data propagated throughout the ecosystem to include the analytics and reports.

DataKitchen helped us completely transform our operations by broadening our definition of testing. Testing the DataKitchen way was not limited to checking for basic attributes such as columns and rows; we expanded testing to include data accuracy and continuity. Tests assess important questions, such as "is the data correct?" When business rules are applied, "does the data make sense?" "Is there something new or unusual," and does the data align with its history and related data sets?" The DataKitchen platform not only makes this level of testing possible but also practical.

The team's increased testing capability led to another best practice. When a new error or anomaly is detected in the data, the platform makes it easy to build a new test allowing the team to catch that same error in the future. The real-world impact of this type of testing cannot be understated. When the team can be confident that 99%+ of data errors will be caught before analytics or reports are published, it gives everyone confidence in the results and builds credibility across the company.

Many teams automate, but DataKitchen inspired a new level of automation in our analytics. We challenged the Celgene team to think in these terms. If I am going to do a task more than twice a year, why wouldn't I automate it? The relatively moderate effort in automation paid massive dividends by allowing team members to always be working on the next important question or searching for the next big insight. Testing and automation alone helped the team take back hours and hours of non-value-added effort each week.

Agile practices also helped to transform the team. We never did Agile just for the sake of doing Agile. Instead, we adopted the mindset of trying many techniques and adopting what worked. For the Celgene team, Scrums had a positive impact, as did working from a backlog. In the case of Scrum, the simple act of being able to adjust efforts daily across the team helped to ensure people were working on the highest priority project at any given time. Working from a backlog helped the team prioritize work without the fear of "dropping the ball" on a request or project. Over time the team developed the practice of sharing the backlog with our partners, who helped us prioritize efforts. Because of this practice, the status of a request was always transparent, and if the work was further down the backlog or removed altogether, everyone understood why.

Over this multi-year journey, the DataKitchen team never lost their drive to help make our team successful. They always delivered exceptional results and taught us how to do the same in the DataOps arena. The team at DataKitchen does have the recipe for success in helping teams spend more time bringing value through a DataOps transformation.

SECTION 4

The process hub and the platform

1. Centralize Your Data Processes With a DataOps Process Hub
2. Large Pharma Achieves 5X Productivity Gain With DataOps Process Hub
3. Implementing a Pharma Data Mesh using DataOps
4. Webinar Summary: Data Mesh and Data Products

Centralize Your Data Processes With a DataOps Process Hub

Read the original article at: <https://datakitchen.io/blog/centralize-your-data-processes-with-a-dataops-process-hub/>



Data organizations often have a mix of **centralized and decentralized** activity. DataOps concerns itself with the complex flow of data across teams, data centers and organizational boundaries. It expands beyond tools and data architecture and views the data organization from the perspective of its **processes and workflows**. **DataKitchen DataOps Automation** is a “*process hub*” that masters and optimizes those processes. The term process hub is new to most people, so let’s look at a concrete example of how a process hub is used in a real-world application. The requirement to integrate enormous quantities and varieties of data coupled with extreme pressure on analytics cycle time has driven the pharmaceutical industry to lead in DataOps adoption.

The Last Mile Problem

During a **pharmaceutical launch** or any phase in a drug’s lifecycle, the data team must utilize processes that promote teamwork and efficiently act upon the data. The bottom line is how to attain analytic agility? The typical pharmaceutical organization faces many challenges which slow down the data team:

- Raw, barely integrated data sets require engineers to perform **manual**, repetitive, error-prone work to create analyst-ready data sets.
- A slow, un-agile approach to supporting commercial pharma analytics means that the team must do data work manually.

- Repetitive data process and query work reduce the time the commercial pharma analytics team has to create original insight.
- A lack of shared and **reusable** ‘ingredients’ extends the time required for analysts to perform ad hoc analyses.
- Firefighting and stress are common.
- **Errors** are caused by inconsistencies in how the analytics team applies data processes and business logic.

Cloud computing has made it much easier to integrate data sets, but that’s only the beginning. Creating a data lake has become much easier, but that’s only ten percent of the job of delivering analytics to users. It often takes months to progress from a **data lake** to the final delivery of insights. One data engineer called it the “last mile problem.”

In our many conversations about data analytics, data engineers, analysts and scientists have verbalized the difficulty of creating analytics in the modern enterprise. These are organizations with world-class data engineers and the industry’s best-in-class toolchains. When we hear articulations like these, we recognize that they have neglected to pay attention to their processes and workflows.

- “How do we execute the same level of service **faster**?”
- “What kind of back-end process can we institute to answer ‘what-if’ questions faster?”
- “My team is being too reactive. We need to be more proactive with our customers.”
- “This is a last-mile problem.”
- “Right now, this is a 90% commercial pharma analytics problem and 10% an IT problem.”
- “It’s too hard to change our IT data product. I don’t understand how they do it or QA it.”
- “Can we create high-quality data in an “answer-ready” format that can address many scenarios, all with minimal keyboarding?”
- “I wish I were in a better position. I get cut off at the knees from a data perspective, and I am getting handed a *sandwich of sorts* and not a good one!”

One DataKitchen customer likes to point out that he feels that analytics teams in commercial pharma are competing with **Amazon**. Not because of AWS or some other technology, but rather because companies like Amazon have compressed response and delivery times. If Amazon can deliver an order in one or two days, business users want the same response time from the analytics team. They want questions answered the next day and answers being delivered to the field next week. It is an on-demand world. How does an analytics team attain that insane velocity of finding quick answers to questions and then cranking out production-grade insights?

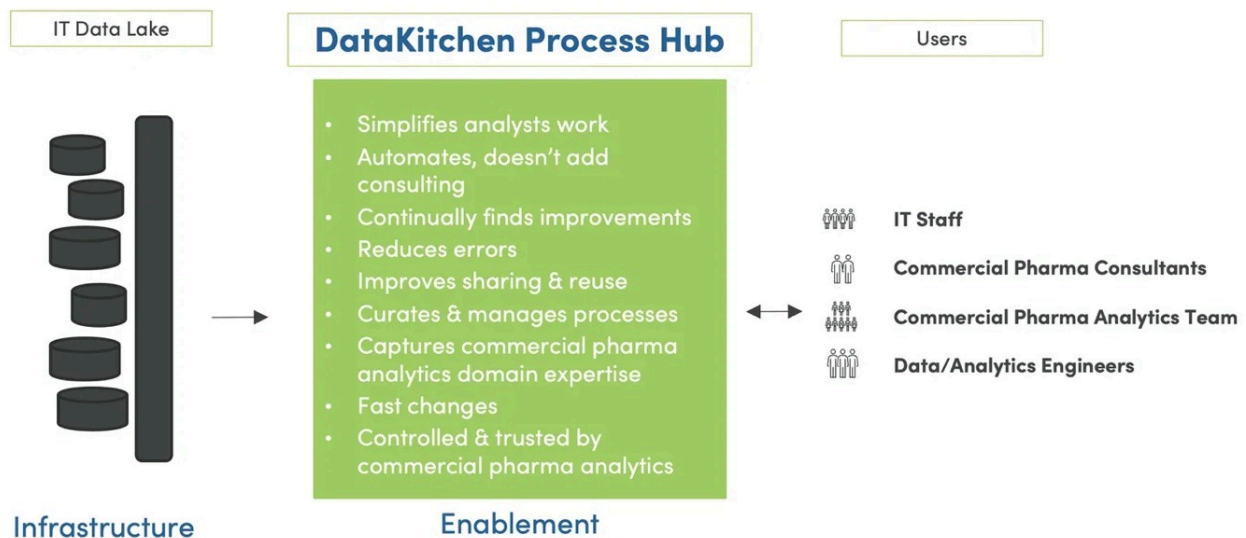


Figure 1: DataKitchen DataOps Automation serves as a process hub.

It's common for organizations to have built a data lake with lots of data - or a data hub to facilitate data exchange. They lack a place to centralize the processes that act upon the data to rapidly answer questions and quickly deploy sustainable, high-quality production insight. Automation provides a way to accomplish this without hiring expensive teams of **consultants**. Think of it as a way to curate and manage the processes that act upon data and gather analytics domain expertise in one controlled and trusted place instead of keeping it on desktops distributed throughout the organization.

Centralize the Process

*If you centralize the process, you can generalize the process. By generalizing the process, you can save time and effort, thereby freeing up the capacity to get more insight to your customers. You can think of DataKitchen DataOps Automation as a "process hub" that **coordinates activity** across multiple teams, vendors and locations with governed, reusable process components that automatically detect errors. DataOps works on top of your existing data, allowing you to use your **favorite tools**, whether you prefer SQL, Alteryx, Tableau, Looker or whatever else. DataOps is an open, **secure** platform that enables anyone granted permission to access data and update the processes and workflows that operate on data. DataOps eliminates **bottlenecks** and delays by putting everything related to analytics under the control of the internal data team and reducing the need for outside consultants.*

DataKitchen DataOps Automation delivers on these capabilities and many more:



- Automated, **governed** and reusable processes
- Multi-team, multi-vendor and multi-location **collaboration**
- Rapid turnaround time
- Automated **error reduction**
- Analyst-ready trusted **data sets**
- **Tools agnosticism** - works on top of an existing data lake and your toolchain of choice
- Reusable libraries of **recipes** and **ingredients**
- Open, secure platform for anyone to: Access data and analytics Change the processes used to create data and analytics
- Access data and analytics
- Change the processes used to create data and analytics

Figure 2: Employing a DataOps Platform as a process hub minimizes the cost for new analytics.

Your IT organization may have a permanent data lake, but data analytics teams need the ability to rapidly create insight from data. DataKitchen DataOps Automation serves as a process hub that builds temporary analytic databases for daily and weekly ad hoc analytics work. These limited-term databases can be generated as needed from automated recipes (orchestrated pipelines and qualification tests) stored and managed within the process hub.

The process hub, in the middle of figure 2 above, is built for rapid change. It is expressly designed to minimize the cycle time of translating an idea into high-quality production insight. DataKitchen DataOps Automation is based on a “*process first*” principle that minimizes the “**cost per question**.” How can the investment required to create new analytics be reduced such that it is possible to ask many more questions in a shorter amount of time? The process hub capability of DataKitchen DataOps Automation ensures that those processes that act upon data - the tests, the recipes - are shareable and manageable.

Example Process Hub for a Biologic Launch

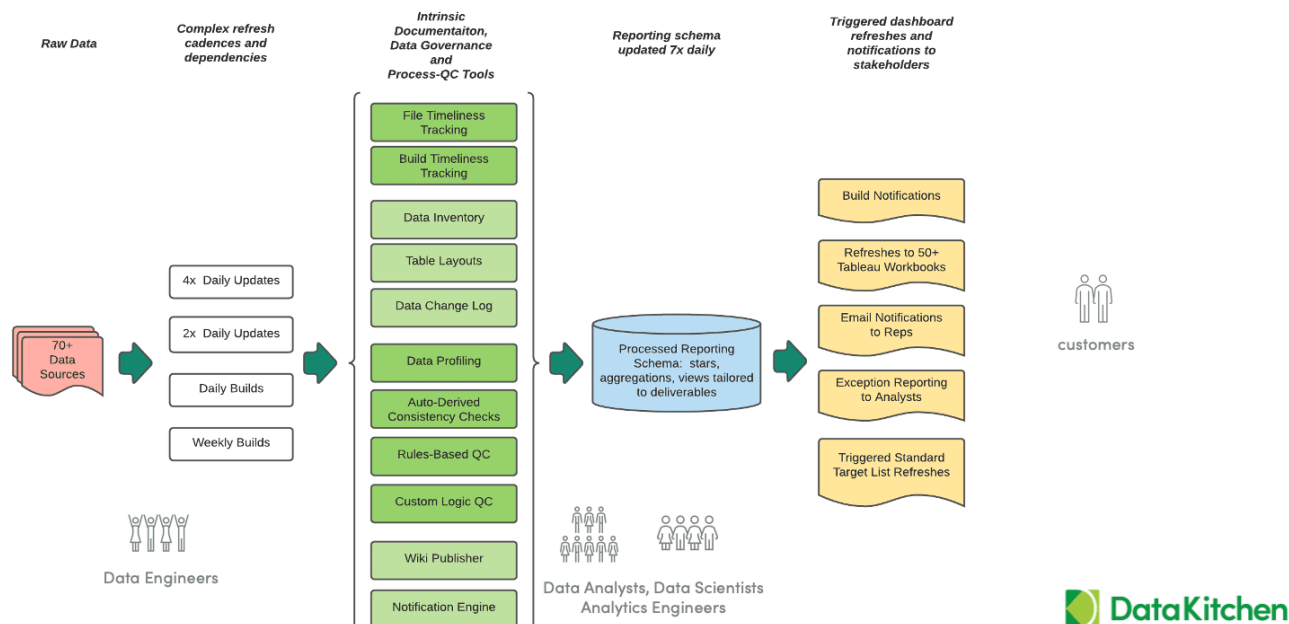


Figure 3: Example process hub for biologic launch

Let's take a look at an example DataOps process hub for an actual biologic launch. The data pipelines must contend with a high level of complexity - over seventy data sources and various cadences, including daily/weekly updates and builds. Biologic data is complicated and not clean, so the process hub first profiles the data and imposes automated, rule-based quality checks. Has the data **arrived on time**? Is the quantity of data correct? When the tests pass, the orchestration admits the data to a **data catalog**.

New data is shared with users by updating reporting schema several times a day. This delivery takes the form of purpose-built data warehouses/marts and other forms of aggregation and **star** views tailored to analyst requirements. When these builds are complete, notifications trigger refreshes of dashboards, Tableau workbooks, and whatever standard reports the business unit requires.

The analytics supported by the DataOps process hub are extremely agile from two perspectives. The production analytics are updated several times a day using dozens of data sources arriving asynchronously. Perhaps more importantly, members of the data team may quickly update process orchestrations related to data at any time. With a process hub, it is possible to make dozens of process changes per week. Deployments include new data sets, new schemas, new views, whatever is required. This flexibility is accomplished while maintaining an extremely low error rate because the process tests data at every stage of the process. Errors are caught by the data team, not the consumers of analytics.

The DataOps process hub implements automation that replaces an army of people who previously executed manual tests, checklists and procedures. The team redeploys its newly freed resources on projects that create analytics that fulfill business requirements. The DataOps process hub allows

commercial pharma analytics teams to produce more insight, faster and at the speed that the customers and business demand. Some features and benefits of DataKitchen DataOps Automation are shown in the table below.

FEATURE	BENEFIT
Process hub that operates on data designed for commercial analytics	Significantly faster time-to-insight for ad hoc requests
Automating (and reusing) process with a button push	Significantly fewer dollars spent on analytics consultants and staff time
Enablement for commercial analytics, not just IT infrastructure	Extend IT investment, not replace

Table 1: DataKitchen DataOps Automation process hub features and benefits.

When the processes that act upon data are designed for rapid refresh and rapid-response to analytics questions, the data team can achieve a much faster time-to-insight and lower cost of implementation for ad hoc requests. DataOps automation replaces the non-value-add work performed by the data team and the outside dollars spent on consultants with an automated framework that executes efficiently and at a high level of quality. Focusing on the processes that operate on data enables the team to automate workflows and build a factory that produces insights.

The DataOps process hub does not replace a data lake or the data hub. It is an automation layer that spans the entire existing toolchain. If you are an Informatica shop or run Snowflake or Databricks, DataKitchen works alongside data ecosystem tools and integrates them into the process hub.

Although this might be seen as heresy by some, the DataOps **process-centric approach** considers the *processes that act upon data are as important as the data itself*. Neglecting those processes opens an organization up to delays, errors and unplanned downtime. Embracing the process-oriented approach to data analytics leads to ever faster and more flexible analytics development and operations with **virtually zero downtime**.

The DataOps Advantage

From a DataOps perspective, success is being able to build analytics consistently across dozens of product teams. It's about deploying pipelines at scale across a large number of people and spanning a multitude of toolchains. It's about quality assurance, automation, reusability, and repeatability. DataOps strives for analytics agility which leads to business agility. When the analytics team works rapidly and accurately, they can do a better job supporting decision-makers. It's that simple.

DataOps is becoming an essential methodology in enterprise data organizations. Companies that implement it will derive significant competitive advantage from their superior ability to manage and create value from data. They will be able to produce high-quality, on-demand insight that consistently leads to successful business decisions. DataOps is fundamentally about eliminating errors, reducing cycle time, building trust and increasing agility. Companies that do these things well will win the next business cycle.

Large Pharma Achieves 5X Productivity Gain With DataOps Process Hub

Read the original article at: <https://datakitchen.io/blog/large-pharma-achieves-5x-productivity-gain-with-dataops-process-hub/>



The Challenge

A large pharmaceutical Business Analytics (BA) team struggled to provide timely analytical insight to its business customers. The company invested significant effort into managing lists of potential prescribers for certain drugs and treatments. However, the BA team spent most of its time overcoming error-prone data and managing fragile and unreliable analytics pipelines.

Customers and market forces drive deadlines and timeframes for analytics deliverables regardless of the level of effort required. It may take six weeks to add a new schema, but the VP may say she needs it for this Friday's strategy summit. The BA team was under constant pressure to deliver analytics on demand.

The BA team often did not receive a data platform tailored to their needs. IT gave the business analysts a bunch of raw tables, flat files and other data. What they received from IT and other data sources were, in reality, poor-quality data in a format that required manual customization. Out of hundreds of tables, business analysts had to pull together the ten tables that mattered to a particular question. Tools like Alteryx helped a little, but business analysts still performed an enormous amount of manual work. When the project scope and schedule were fixed, managers had to add more staff to keep up with the workload.

If the IT or data engineering team did not respond with an enabling data platform in the required time frame, the business analyst performed the necessary data work themselves. This ad hoc data engineering work often meant coping with numerous data tables and diverse data sets using Alteryx, SQL, Excel or similar tools. Business analysts were living under constant pressure to deliver. They spent 80% of their time performing tasks that were necessary but far removed from the insight generation and innovation that their business customers needed.

The dynamic nature of data leads to complexity, including significant quality control (QC) challenges. Testing and validating analytics took as long or longer than creating the analytics. QC is extraordinarily time-consuming unless it is automated. Business analysts needed the flexibility that comes with control of data and orchestrations. Management of the process can't be performed "off the grid" without control, enforcement and consistency.

The Solution

To meet the demand for continuous, high-quality insight, the BA team implemented a DataOps "process hub." This allows the Data Analysts to take control of the workflows that operate on and around data. A process hub ensures that the processes and workflows that create the enterprise data platform are just as important as the data itself. Some data analysts use long checklists to execute steps that integrate, clean and transform data. A process hub automates, synthesizes and synchronizes these steps, so they do not have to be executed and managed manually.

Workflows that add value to data are critical elements of intellectual property that must be retained and controlled. A DataOps process hub instantiates all processes and workflows (i.e., tribal knowledge) related to data and analytics into code. With a process hub, workflows are curated, managed and continuously improved. A robust and vibrant process hub is a clear sign of data organizational maturity.

The process hub manages significant complexity because data flows in many directions between platforms, tools, people and functions. As data keeps changing, the process hub enforces quality and the functional order of operations. Valid data depends on a robust, repeatable process. The process hub catches and manages data and analytics errors. Without the process hub, errors cause outages. There is no "single version of the truth" without a "single version of the process."

This large enterprise has many products and brands with overlapping marketing campaigns. The process hub leverages the work product of several operational systems depicted on the right side of figure 1: a master data management (MDM) system, opt-out management from active marketing campaigns, a CRM and a marketing campaign management platform. These systems interface to the process hub through automated orchestrations. If data is sequestered in access-controlled data islands, the process hub can enable access. Operational systems may be configured with live orchestrated feeds flowing into a data lake under the control of business analysts and other self-service users.

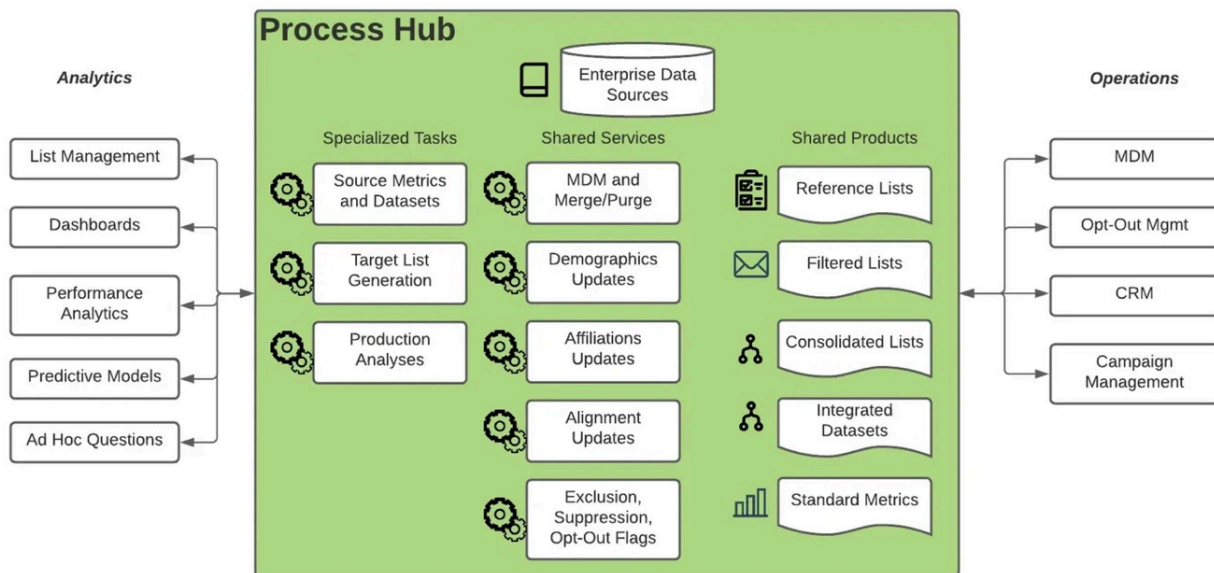


Figure 1: A DataOps Process Hub

The process hub provides shared services and resources that feed the needs of the enterprise, divisions, product teams and specific projects while also providing feedback to operational systems. On the left side of figure 1, we see analytics produced for the various business development stakeholders. Despite the complexity, mission-critical analytics must be delivered error-free under intense deadline pressure.

The business analysts creating analytics use the process hub to calculate metrics, segment/filter lists, perform predictive modeling, “what if” analysis and other experimentation. Requirements continually change. Data is not static. Data workflows are not like IT infrastructure that is set up and then simply maintained. New analytics are added, and existing analytics are refined. The process hub resynchronizes master data, updates demographic information and reapplies suppressions, opt-outs and other attributes.

The Results

With the DataOps process hub in place, the BA team is significantly more productive. The process hub lowers the cost per additional data and analytics question. It centralizes the process so that it can be generalized and then applied to specific analytics questions in a fraction of the time it would otherwise require. An automated, governed, and reusable process decreases unplanned work and reduces the cycle time of change. It also minimizes the need for outside consultants who tend to rely upon heroism and tribal knowledge. A process hub is the key to creating an open, secure platform that enables those with access to efficiently update and skillfully manage the processes used to create data and analytics.

Bottom line, the process hub enabled the company to deliver higher quality analytics at a faster rate with fewer business analysts. They experienced:

- Delivery of new analytics in one-third the time;
- Data errors reduced by 80%;

- 5X increase in the number of sales and marketing users supported per data analyst; and
- Visualizations updated per week increased from 50 to 1500

The process hub helped this pharmaceutical company achieve its strategic goals by improving analytics quality, responsiveness, and efficiency. The process hub supports improved processes, automation of tools, and agile development of new analytics. With a process hub, the analytics team delivered value to users in record time, accelerating and magnifying their impact on top-line growth.

For more on process hubs, please see our post titled “[DataOps For Business Analytics Teams.](#)”

Implementing a Pharma Data Mesh using DataOps

Read the original article at: <https://datakitchen.io/blog/implementing-a-pharma-data-mesh-using-dataops/>



Below is our fourth post (4 of 5) on combining **data mesh** with **DataOps** to foster innovation while addressing the challenges of a decentralized architecture. We've covered the basic ideas behind data mesh and some of the difficulties that must be managed. Below is a discussion of a data mesh implementation in the pharmaceutical space.

For those embarking on the data mesh journey, it may be helpful to discuss a real-world example and the lessons learned from an actual data mesh implementation. DataKitchen has extensive experience using the data mesh design pattern with pharmaceutical company data.

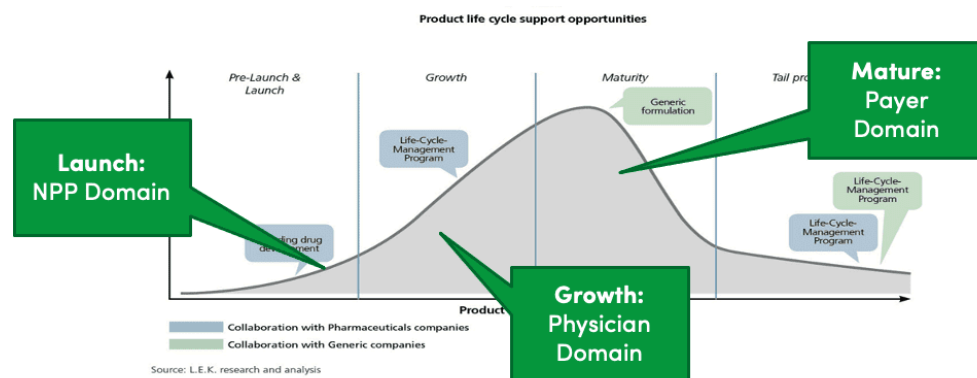
In the United States, private manufacturers of pharmaceuticals receive a patent for a limited period of time - approximately 20 years. In figure 1 below, we see that the data requirements are quite different for each of three critical phases of a drug's lifecycle:

Lifecycle Phase	Marketing Target	Data Focus
Launch	Patients	Non-Personal Promotion: emails, website visits, even radio ads
Growth	Physicians	Sales, claims data, anonymized patient data
Mature	Payer / Insurance Co.	Rebates, formulary

Table 1: Lifecycle phases of pharmaceutical product launch

Each distinct phase of the drug lifecycle requires a unique focus for analytics. During the launch phase, the focus is on marketing to patients through consumer channels. A successful launch lays the groundwork for the growth phase where (the manufacturer hopes) physicians prescribe the drug. As generic alternatives become available, the market enters the maturity phase where cost efficiency and margins become most important. There are different teams within the pharmaceutical company that focus on the respective target markets. Each phase is important, but a weak launch cuts the legs out from under the growth phase and decreases the lifetime revenue of the product. The analytics team is under tremendous pressure during the early phases of the drug's lifetime. The business users propose questions and ideas for new analytics and require rapid response time. Data mesh and DataOps provide the organization, enterprise architecture, and workflow automation that together enable a relatively small data team to address the analytics needs of hundreds of active business users.

Commercial Pharma Analytics



US Commercial Pharma Domains

- **NPP** (Non-Personal Promotion): emails, web site visits, even radio ads
- **Physician**: doctor (& other outlets) sales, claims data, anonymized patient data
- **Payer**: Payer/Plan, rebates, formulary

Figure 1: Data requirements for phases of the drug product lifecycle

Pharma Data Requirements

Drug companies maintain internal data sets and rely heavily upon third-party data (a multi-billion dollar industry). Some data sets are used by multiple teams, but that introduces complexity. Figure 2 shows the vast array of data sets used by pharmaceutical product launch organizations.

What About the Data?

Sub-national Weekly data
 Sub-national Payer Data
 Sub-national Institutional (DDD) Data
 National Prescription Audit Data
 Sales Force Alignment Data
 Longitudinal Patient Data
 Sub-national Profit and Loss Data
 Sub-national Claims and Co-pay Data
 Payer and Plan Formulary Data
 Census Data
 Stocking Data
 Source of Business
 AMA Data
 Retail OTC Data
 Buy and Bill Data
 Field Calls and Promotional Activity Data
 Rep Expenses and Vacancy Data
 Hotline Verification Data
 Contract and Payer Rebates Data
 Veeva CRM Data
 ERP Data
 NPP Data
 Forecast Data
 Primary Research Data

What about the data in each domain?

- Each domain has separate data sources
- Overlapping entities (e.g., physicians) exist in each domain
- Each domain has different cycle times of product (i.e., daily, weekly, hourly, etc.)
- Each data domain has its unique characteristics.
- For instance, subnational physician data from IQVIA - purchased by pharma companies - may not 1:1 match claims data, which may not match payer data. This is due to data supplier issues & timing projection algorithms.

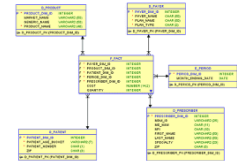


Figure 2: Data feeding the drug product lifecycle domains

Some data sets are a mix of actual and projected data, complicating their use with other data sets that purport to be the same but use a different algorithm to fill in gaps or posit projections. Two data sets of physicians may not match. They each tell a different story about the data.

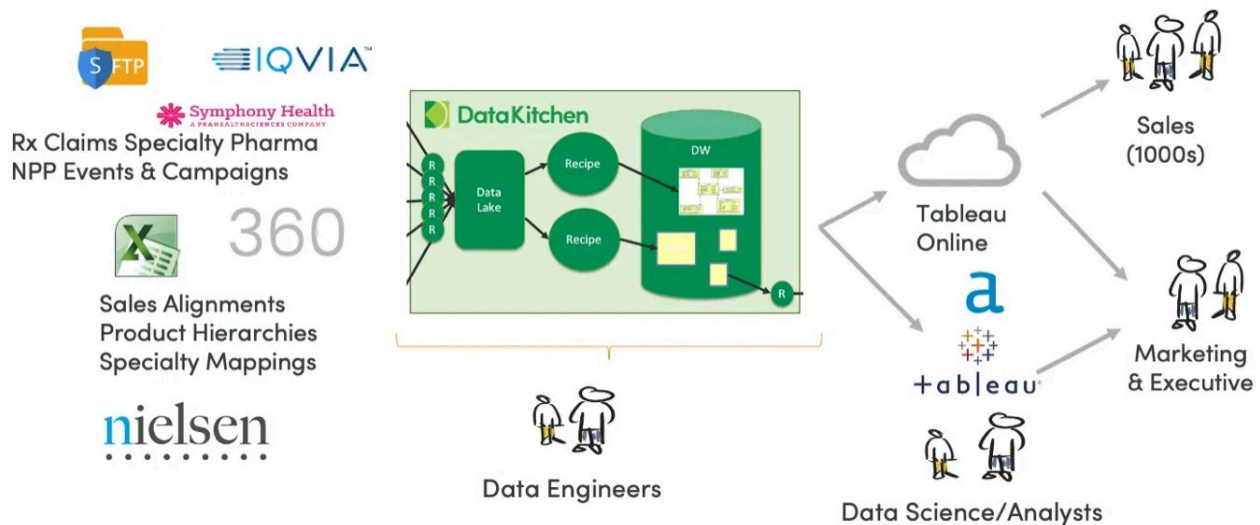


Figure 3: Example DataOps architecture based on DataKitchen DataOps Automation

Figure 3 shows an example processing architecture with data flowing in from internal and external sources. Each data source is updated on its own schedule, for example, daily, weekly or monthly. DataKitchen DataOps Automation ingests data into a **data lake** and runs **Recipes** to create a data

warehouse leveraged by users and self-service data analysts. These users are very important to the company because their success greatly influences the strength of the revenue ramp in the drug's growth and maturity phases.

A sales or marketing team member could propose an idea - - what if we combined data from sources A and B to find potential customers for our new product? The data engineer updates the Recipe (orchestration) that feeds the data lake if a data source needs to be added and modifies the Recipe that generates the data warehouse. The new Recipes run, and BOOM! The data scientists and analysts have what they need to build analytics for the user. With DataOps automation, making an analytics change is cut to a fraction of what it takes to execute an update manually.

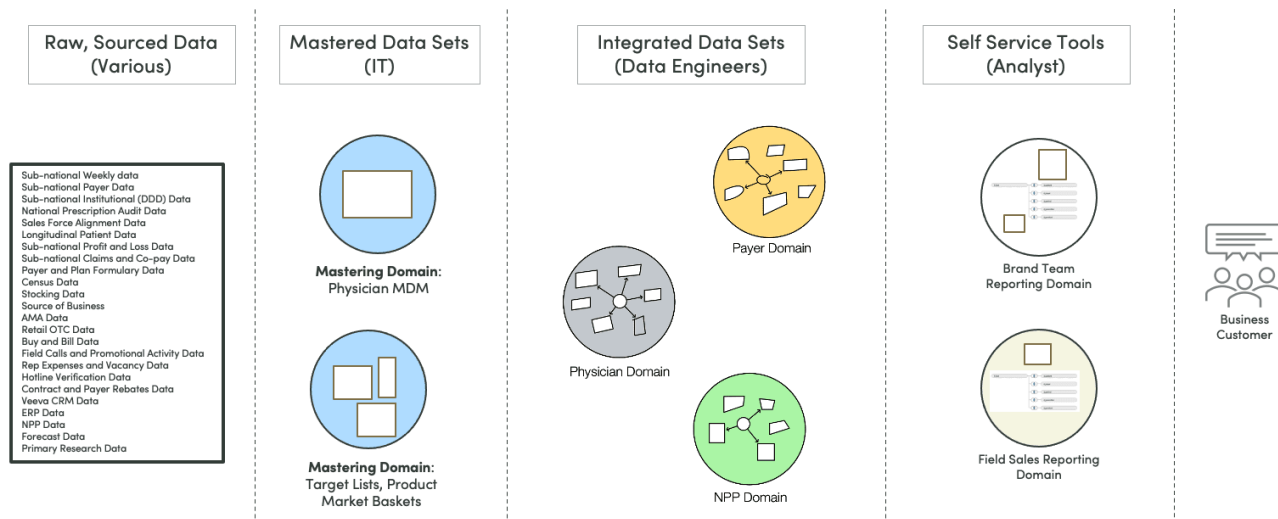


Figure 4: Example domain partitioning

Pharma Data Mesh Domains

Placing responsibility for all the data sets on one data engineering team creates bottlenecks. Let's consider how to break up our architecture into data mesh domains. In figure 4, we see our raw data shown on the left. First, the data is mastered, usually by a centralized data engineering team or IT. Each of the mastered data sets could be a domain. For example, there may be one million physicians in the US, but for a given drug product, perhaps only 40,000 are important. Getting this standardized is vital because it affects sales compensation.

The second set of domains are the integrated data sets created by data engineers in the form of data warehouses or analytic data marts. There are facts and dimensions, along with multiple tables, used to answer business questions. In many cases, these are star schemas.

The third set of domains are cached data sets (e.g., tableau extract) or small data sets that self-service analysts can mix with the central data in Alteryx or other tools. Self-service data science teams may require their own segmentation models for building reports, views, and PowerPoints.

Each circle in figure 4 could be its own data mesh domain. The separation of these data sets enables the teams to decouple their timing from each other. In figure 5, we see that each domain has its own domain update processing (**Recipes** or data analytics pipelines), represented by a **directed-acyclic graph** (DAG). For each domain, one would want to know that a build was completed, that tests were applied and passed, and that data flowing through the system is correct.

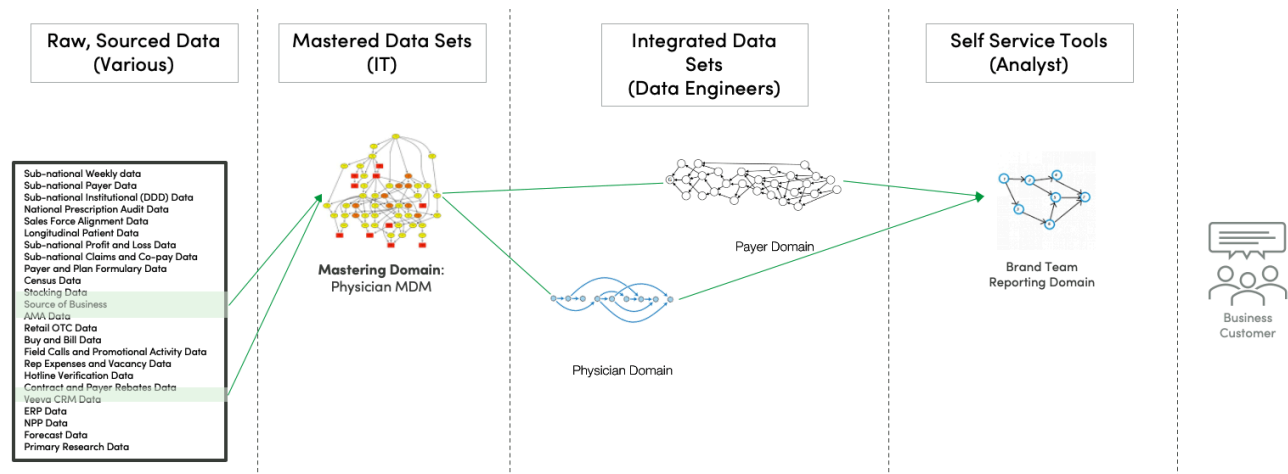


Figure 5: Domain layer processing steps

There's a top-level DAG relationship between the raw data, arriving asynchronously, the mastered domain, the integrated data sets and the self-service domain. One challenge is that each domain team can choose a different toolset that complicates multi-level orchestration, testing and monitoring. A DataOps superstructure like DataKitchen that supports connectors to data ecosystem tools addresses this issue.

Another challenge is how to manage ordered data dependencies. The American Medical Association (AMA) may update its data set of physicians, which flows into the physician mastered domain. It then gets used by the physician and payer data warehouses which are eventually used by the self-service teams. We would not want the physician data set in the physician domain and the payer domain to drift apart or get "out of sync," which might happen if they are updated on different iteration cadences. While the goal of a data mesh is to empower teams through decentralization, the overall architecture has to consider the order-of-operations dependencies between the domains. Manual processes that enforce synchronization and alignment interfere with domain-team productivity and are a common cause of unhappy users.

To manage complexity, the system requires inter-domain communication shown in the table below. A domain query provides information about builds, data, artifacts, and test results. Process linkage aids in multi-level DAG orchestration. Some designs perform process linkage with an event bus that marks the completion of a DAG by putting an event on a Kafka queue, using a publish/subscribe model. Data linkage refers to the sharing of common tables or the output of data from one domain being fed into another. Finally, there is development linkage. Can development environments with all related domains be created easily? Can they be modified? Is there a seamless path to production?

Inter-Domain Communication	Question / Steps Asked
Domain Query	• “When was the last time you were updated?” Successful or failure? Warnings? • “Is the data or artifacts in your domain good? Can you prove it with some test results?”
Process Linkage	• “Ok, you start. I am done.” • “Ok, you start. I am done <i>AND</i> here are a bunch of parameters you need to keep going.”
Event Linkage	“Here is an event: e.g., processing completed, error, warnings, etc.”
Data Linkage	“We share a common table (e.g., a dimension table) in our domain.”
Development Linkage	• “Can I re-create your domain in development?” • “Can I see the code you used to create it?” • “Can I modify that code in development?” • “Is there a path to production?”

Table 2: Inter-Domain Communication

The many ports or access points of inter-domain communication illustrate why it’s so helpful to support URL-based queries. It enables the data engineer to ask questions, parse responses and set up automated orchestrations instead of maintaining checklists and manual procedures that someone must execute.

Conclusion

While there are unique aspects to the pharmaceutical industry, the flow from data sources to a data lake to a data warehouse to self-service analysts is a common design pattern that cuts across industries. Data mesh organizes teams and the system architecture so that mastered data sets, integrated data sets and self-service users can work independently and together. DataOps automates processes and workflows so that the productivity of any individual domain team isn’t overwhelmed by the complexity of the overall system. DataOps keeps inter-domain tasks flowing so that the domain team can progress at their optimal speed.

Data mesh is a powerful design pattern that leading enterprises are using to organize their enterprise analytics architectures. In our next post, we will look at tools capabilities essential for building a successful data mesh. Our discussion focuses on the value of automation and how DataOps serves as the wind beneath the wings of data mesh. If you joined this series midway, you can return to the first post here: ” [What is a Data Mesh?](#)”

Webinar Summary: Data Mesh and Data Products

Read the original article at: <https://datakitchen.io/blog/data-mesh-and-data-products/>



Chris Bergh, CEO of DataKitchen, [delivered a webinar on two themes - Data Products and Data Mesh](#). Bergh started by discussing the complexity within data and analytics teams, stating that complexity makes everything more complicated and, in the long run, it kills productivity. Examples of complexity within data and analytics teams include multiple teams with different roles, thousands of different data sets of different types and shapes, each with unique needs, and interconnected project plans.



Bergh went on to talk about how the software industry has tackled complexity by applying lean and agile principles such as DevOps and domain-driven design software products. The data industry is now adopting similar principles, such as data testing instead of test-driven development, data observability instead of observability, and functional data engineering instead of functional programming.

From Software Engineering to Data Analytics

<u>Software Engineering</u>	<u>Data & Analytics</u>
• DevOps	-> DataOps
• Domain-Driven Design • Software Products	-> Data Mesh -> Data Products
• Test-driven Development • Observability • Event-Driven Architecture • API • Functional Programming	-> Data Testing -> Data Observability -> Streaming; Kappa Architecture -> Data Contract -> Functional Data Engineering

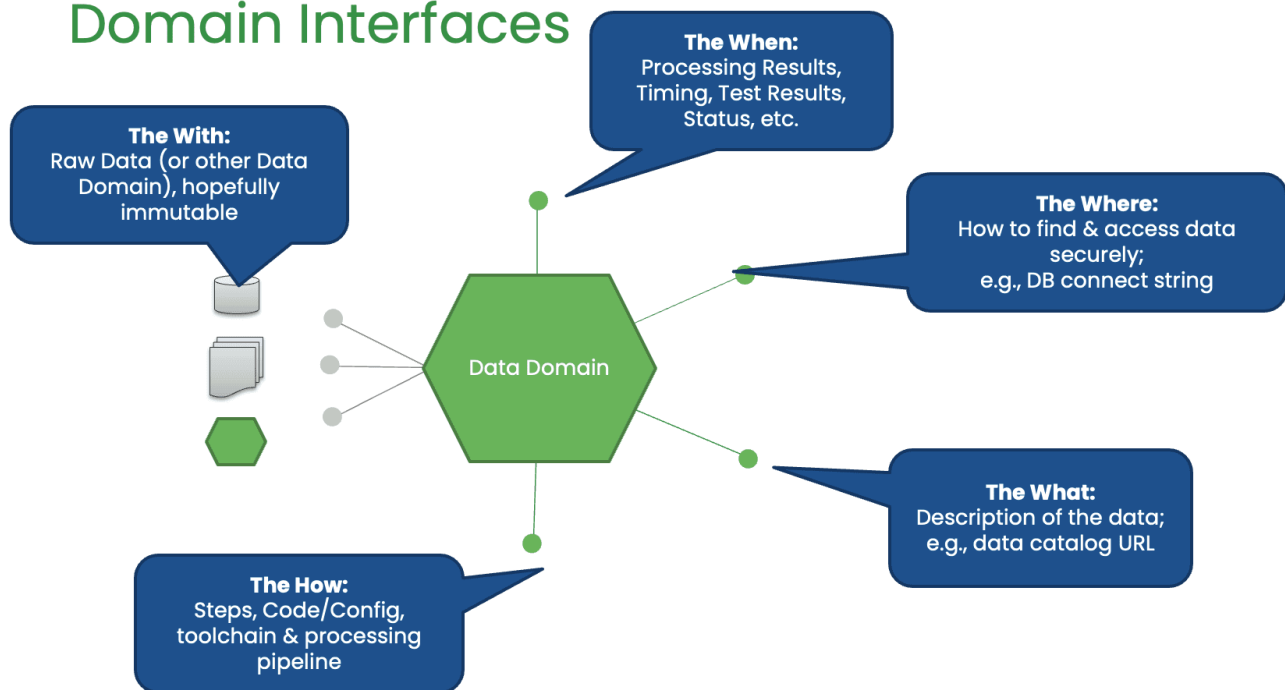
Data Mesh

Bergh explained that the Data Mesh organizes a team's work into chunks called decentralized domains. Instead of boiling the ocean and focusing on all datasets and customers, the Data Mesh focuses on fewer datasets and customers, which reduces complexity and helps get more done.

On the other hand, Data Products are part of an ever-improving flow of value, where success is not defined by completing tasks on a project plan but by delivering insights to customers. Bergh explained that Data Products are about the flow of value to customers, repeatedly and improving the tools, data, and code that make up the product.

Bergh emphasized that Data Products and Data Mesh are not mutually exclusive, and both should be used together to help reduce complexity, increase productivity, and deliver value to customers.

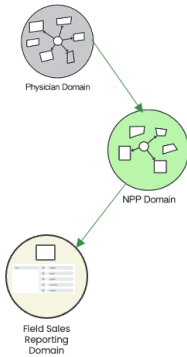
Domain Interfaces



Chris talks about the idea of a 'domain' as a principle of Data Mesh. A domain is a unit that includes integrated or raw data, artifacts created from data, the code that acts upon the data, the team responsible for the data, and metadata such as data catalog, lineage, and processing history. The goal of a domain is to deliver value to the customer, which may be another team or a business user. The speaker emphasizes the importance of independence, trust, usability, discoverability, security, and interoperability in a domain. They suggest that a domain architecture can clarify responsibilities and improve customer intimacy. They describe five interfaces to a domain: the width (data), the where (location), the what (description), the how (process), and the who (team).

He also highlights the composability of domains, meaning that the work in one domain can be the input of another domain. This allows teams to control their own world but also utilize other people's work. Chris argued that focusing on services provided to the customer rather than the project plan can improve productivity and value delivery. This approach requires observability to ensure that the domains work well together. Chris discussed the idea of decentralization and how well-designed domain interfaces can provide the freedom to innovate in data engineering. He argued that having central control over data can result in high context-switching costs and hinder innovation.

Inter-Domain Communication Links

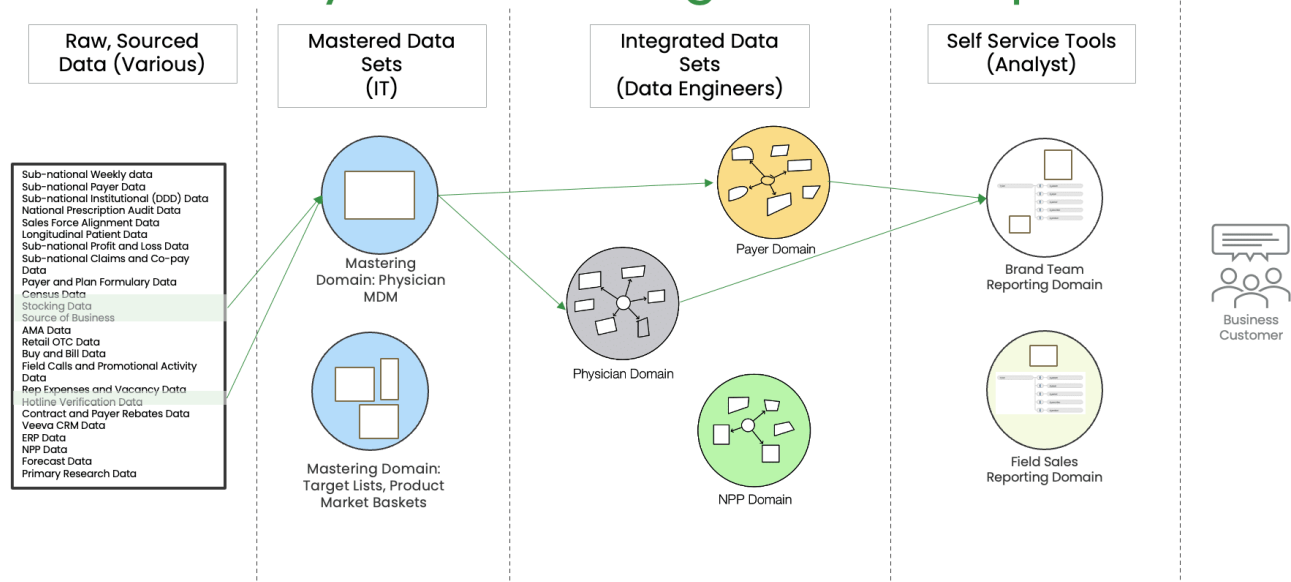


Inter-Domain Communication		Question / Steps Asked
	Domain Query	"When was the last time you were updated?" Successful or failure? Warnings?
	Domain Query	"Is the data or artifacts in your domain good?" Can you prove it with some test results?"
	Process Linkage	"Ok, you start. I am done."
	Process Linkage	"Ok, you start. I am done & here are a bunch of parameters you need to keep going."
	Event Linkage	"Here is an event: e.g., processing completed, error, warnings, etc."
	Data Linkage	"We share a common table (e.g., a dimension table) in our domain."
	Development Linkage	"Can I re-create your domain in development?" "Can I see the code you used to create it?" "Can I modify that code in development?" "Is there a path to production?"

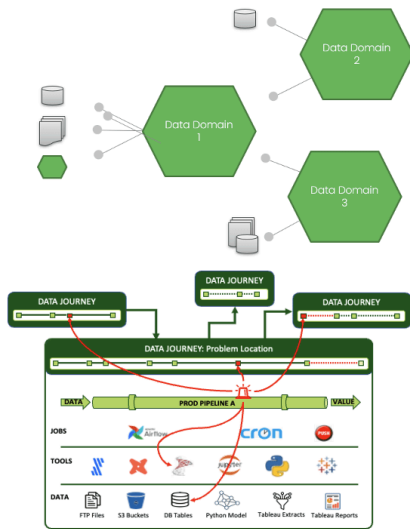
A Data Mesh Example

To illustrate this point, Bergh gave an example of using a Data Mesh in commercial pharma analytics. He explained how different parts of the pharma product lifecycle grow at different times, creating a combination of data and use cases reflected in the teams working on them. The data sets used to understand the launch of a product come from various sources; some are purchased through syndication, some are internal, and some are freely available. Bergh described how the purchase and integration of these data sets could be complicated as they do not come at the same time, have different frequencies, and have different characteristics.

Domain Layers Processing Relationships



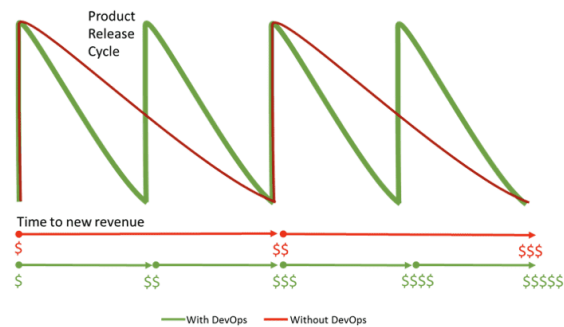
Composition of Domains Need Data Journey observability



- ## Data Products

Bergh suggests that instead of focusing on project plans, data engineers should concentrate on the flow of change to the customer. In contrast to a project-based approach, a product-focused approach is a continuous stream of value to the customer. Iterative development, focus on the flow of value to the customers, and small features delivered quickly to get feedback from the business are crucial components of the product methodology.

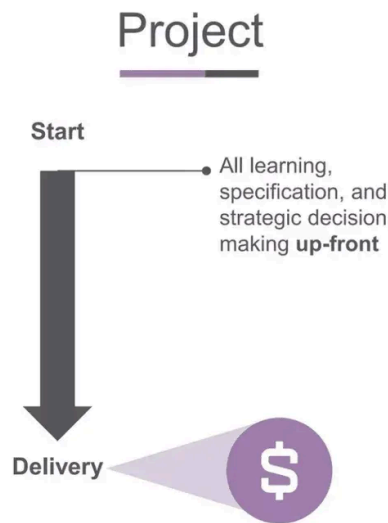
Data Products Are Never Done



- Faster releases of analytics products
- Increased learning of your team
- Better insight
- Faster and more value for your data analytics

The speaker contrasts the fungibility of employees in a project-based approach with a more product-focused approach, where work is brought to people who know the product and customer. Small, cross-functional teams working on a product can learn and innovate more effectively than people working on individual tasks across different projects.

Bergh argues that data projects often involve a lot of waste in terms of waiting or context switching between different roles or projects. The traditional waterfall process, with its wait times between design, defining, development, QA, and release, causes significant delays in delivering the product to the customer. In contrast, the product-oriented approach minimizes the waiting time by having a more frequent checkpoint system and reallocating funding from project to product.



Bergh suggests that a product-based approach to data engineering is more customer-centric than a project-based approach. Success is measured by completing tasks on time and on budget and by how much value the customer has received from the product. The success of a data product is measured by whether the customer is happy with it, whether it has provided an ROI, and whether it has fulfilled its intended purpose.

Overall, Bergh's talk encourages data engineering teams to focus on creating Data Products that provide ongoing value to the customer rather than merely completing tasks in a project plan. The product-oriented approach emphasizes customer-centricity, iterative development, and cross-functional teams that are continually learning and innovating.

SECTION 5

AI and technology

1. The Equation For AI Success: $DT + DX + CTX = 10x$
2. We've Been Using FITT Data Architecture For Many Years, And Honestly, We Can Never Go Back
3. DataOps + FITT + Data Testing = 10x Data Engineering Productivity with AI
4. Build a Data Mesh Factory with DataOps

The Equation For AI Success: DT + DX + CTX = 10x

Read the original article at: <https://datakitchen.io/blog/the-equation-for-ai-success-dt-dx-ctx-10x/>



Everyone wants AI to answer business questions with their data. The reality is most AI assistants built on enterprise data give mediocre, unreliable, or outright wrong answers , not because the AI is bad, but because the foundation underneath it is.

Your AI chatbot is lying to you. Not maliciously. Confidently. It queries your data warehouse, assembles an answer, and presents it with the same tone whether the underlying data is fresh, tested, and correctly joined, or three days stale, never validated, and pulled from a deprecated table nobody deleted in 2021.

This is the AI-on-data problem nobody wants to say out loud: the model is fine. The data underneath it is the disaster. Every hallucination, every wrong answer, every time an analyst says “I don’t trust what the chatbot told me” , that is a data infrastructure failure dressed up as an AI failure.

**Everyone wants
AI to answer
business
questions with
their data.**

**Most AI assistants
give mediocre or
wrong answers.
Not because the AI
is bad, but because
the foundation
underneath it is.**

**The gap is a data problem. Not a model problem.
AI models are amplifiers—if you feed them untrustworthy data,
they amplify bad answers at scale.**

Everyone assumes AI fails at data analysis because the models are not smart enough. The [MIT BEAVER study \(September 2024\)](#) decisively disproves that assumption. When researchers tested leading LLMs against real enterprise data warehouses, models that had scored above 85% accuracy on standard benchmarks collapsed to nearly zero end-to-end execution accuracy on actual business data. The model did not get dumber. The context disappeared. Enterprise schemas were too complex, business questions required joins across many tables and nested aggregations, and the models had never encountered the private data they were suddenly being asked to reason about. When researchers simply handed the models the correct table and column context, performance recovered sharply. The intelligence was there the whole time. What was missing was the foundation underneath it.

[OpenAI's look inside its internal data agent \(January 2026\)](#) reveals something subtle but important: raw data alone is not enough to produce useful insights. Their teams do not just query tables; they ask questions that are grounded in context, such as product launches, feature changes, and user behavior patterns, and the system uses that surrounding knowledge to interpret results. The same metric can mean very different things depending on what was happening in the business at the time, which is why simple SQL outputs often fall short. By combining data with context, their approach turns analysis into something closer to reasoning rather than reporting. The result is that the same data that once produced ambiguity now produces insight, dramatically reducing errors across thousands of employees working on tens of thousands of datasets. It highlights a fundamental truth about data engineering: raw data alone is not enough. The intelligence was already there, but without context, it was incomplete. What was missing was the foundation that connects data to meaning.

A comprehensive [survey of LLM-based text-to-SQL systems published on arXiv \(updated through late 2025\)](#) finds that generating accurate SQL from natural language remains a long-standing challenge, specifically because of the complexities involved in database schema comprehension, and that the problem compounds as databases grow more complex. A separate [ACM Computing Surveys paper on LLMs for text-to-SQL](#) tasks found that domain-specific knowledge is critical, and that performance

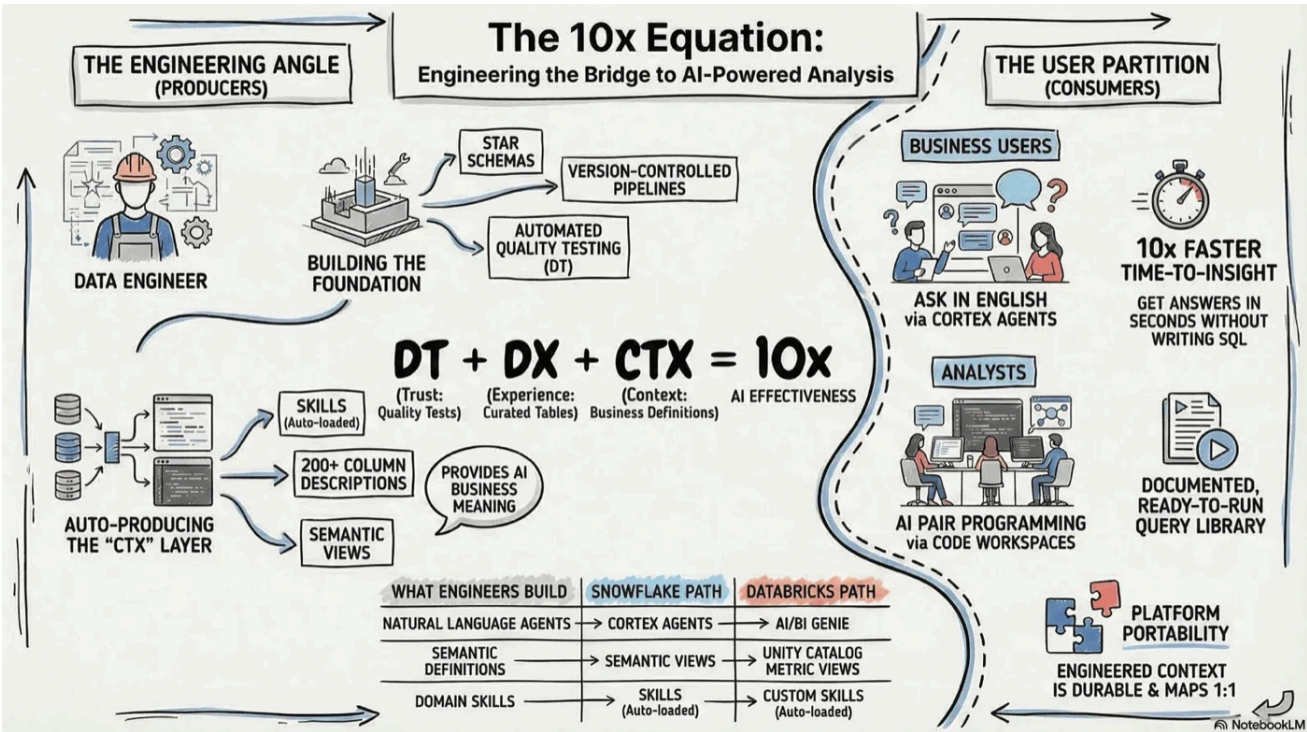
degrades significantly without additional SQL knowledge and external commonsense context. A [2025 arXiv paper on long-context LLMs for natural language to SQL](#) demonstrated empirically that additional contextual information about column references, user-provided hints for clarification, and schema metadata all directly improve execution accuracy, with smaller models benefiting the most from richer context. A [2025 survey from the Journal of the American Statistical Association on LLM-based data agents](#) echoes this theme, noting that LLM-powered data science agents have shown significant potential to transform data analysis, but their role depends on how well the system simplifies complex data tasks for users without deep expertise in the underlying schemas.

The gap comes down to three things. Your team does not fully trust the data the AI is querying. The schema the AI navigates is a graveyard of 150 tables, half of them wrong for the question being asked. And the AI has no idea what your business actually means by “equalized prescriptions” or “territory alignment.” It is a guess. It is always guessing.

Fix those three things, and the picture changes. Not incrementally. Fundamentally. We call it the 10x equation:

DT + DX + CTX = 10x Faster Data Analysis With AI

Data Trust. Data Experience. Context. This is not a theory. It is a checklist for the infrastructure work that makes AI analysis reliable, accurate, and genuinely useful. This is our experience with customers solving this challenge. Let’s break it down.



DT , Data Trust

The first term in the equation is the most foundational: your team must be able to trust the data before AI can use it reliably. Data Trust is not about having perfect data , it's about having tested, monitored, and understood data, delivered to the right people at the right time.

DT: The Foundation of Reliability

Data Trust is not about perfect data—it's about having tested, monitored, and understood data.



What Data Trust Means in Practice

Data Trust has four components that work together:

- **Quality & Testing:** Data quality and testing , Every critical dataset has automated tests validating completeness, freshness, referential integrity, value ranges, and statistical distributions. If a column is supposed to have non-null values, there is a test that fails loudly when it doesn't.
- **Timeliness:** Arrival time and delivery logistics , The right data arrives on time. Late data is wrong data. If the AI is answering questions about yesterday's sales using 3-day-old data, every answer is wrong, regardless of the model's quality.
- **Team Trust:** Team trust , Humans using AI outputs need confidence that the underlying data is reliable. This is cultural as much as technical. A team that has burned bad data will second-guess every AI answer , often correctly.
- **The Intersection:** Right data, right quality, right time , Data Trust is the intersection of all three: the right dataset, verified to an acceptable quality threshold, available when the business needs it.

Why This Is the Foundation

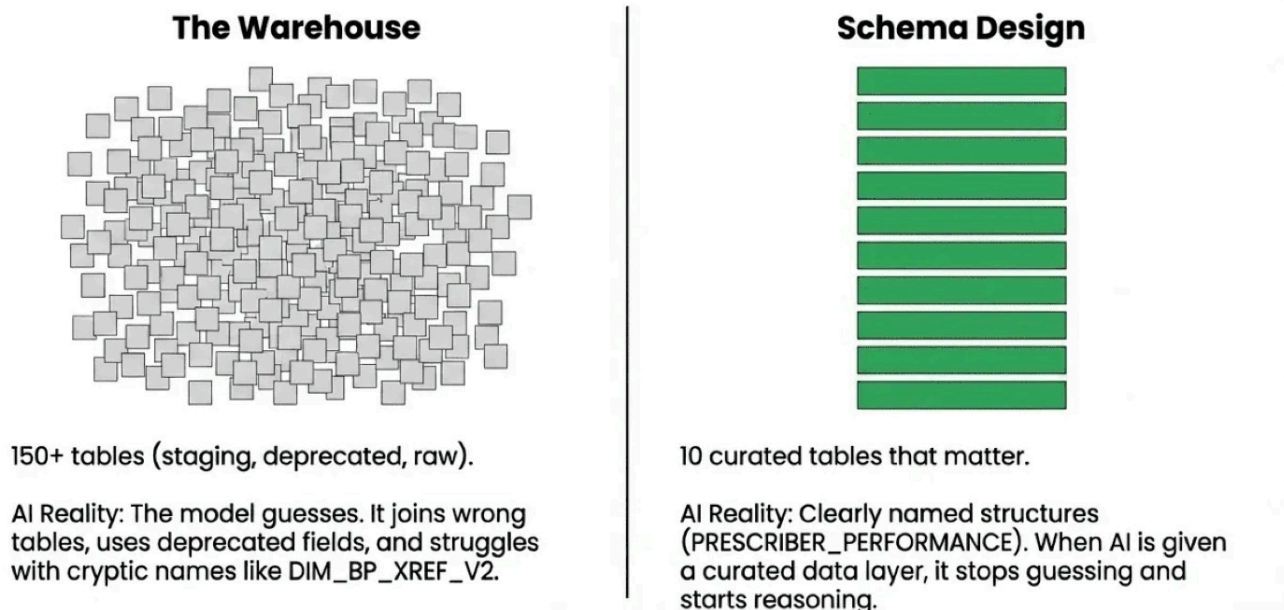
AI models are amplifiers. They make whatever is underneath them faster and more accessible. If what is underneath them is untrustworthy data, AI amplifies bad answers at scale and speed. Data Trust is the prerequisite for everything else in the equation.

Tools like TestGen provide over 120 AI-driven data quality tests that can be applied to any dataset. They automatically profile schemas, flag anomalies, and surface data issues before they reach analysts or AI systems. This foundation is the infrastructure of Data Trust.

DX , Data Experience

Even if your data is trustworthy, AI and analysts alike will struggle if the experience of working with that data is confusing. Data Experience (DX) is about making data legible , a schema design that humans and AI can both read, understand, and use without becoming database archaeologists.

DX: The Hundred-Table Problem



The Hundred-Table Problem

Most data warehouses and lakehouses have accumulated tables over years of work: staging tables, intermediate tables, deprecated views, experimental datasets, and duplicates from three different ETL eras. Nobody deletes anything.

When an AI assistant is given access to a data environment with 150 tables, several bad things happen: The model has to guess which tables are relevant to any given question. It will frequently choose the wrong option, joining tables that are technically related but semantically incorrect. Even when it chooses right, it may use deprecated fields, wrong grain, or mismatched time zones. Performance degrades because the AI explores a vast, confusing space rather than a focused, curated one.

The Solution: Ten Tables That Matter

A good data experience means curating the data landscape. Instead of exposing every table in your warehouse, you build a semantic layer, ten (or twenty, or thirty) clearly named, well-structured tables that represent the core business concepts an analyst or AI needs.

This is not about hiding data. It is about creating a front door. The warehouse can have 150 tables; the AI and the analyst should see the 10 that matter for their domain, with names and structures that make sense.

Good Data Experience looks like: Table names that say what they contain:

PRESCRIBER_PERFORMANCE, not DIM_HCP_XREF_V2. Columns named for business concepts:

EQUALIZED_PRESCRIPTIONS, not EQ_TRX. Consistent grain and join keys are documented explicitly.

No ambiguous foreign keys, if there are multiple ways to join two tables, the right way is marked.

When AI is given a well-structured, curated data layer, it stops guessing and starts reasoning. That is a qualitative shift in output quality.

CTX , Context

Context is the multiplier. It is the difference between an AI assistant that can technically query your database and one that actually understands your business. Context (CTX) is data about data, the accumulated knowledge that turns a generic language model into a domain expert.

CTX: Why AI Needs Context, Not Just Data

Without context, the AI sees:

```
PRODGRP_NO VARCHAR(9)
HCP_ID NUMBER
EQ_TRX NUMBER(38,6)
```

Column names with no business meaning

With context, the AI knows:

- PRODGRP_NO = Product Group Number -- join through NDC instead for accurate product mapping
- HCP_ID = Business Partner ID -- this is the prescriber; join to D_PRESCRIBER for names and territories
- EQ_TRX = Equalized Total Prescriptions -- normalized for days supply; this is the primary volume metric

Context turns a generic chatbot into a pharma analytics expert.

Why AI Needs Context, Not Just Data

Here is the core problem. When you connect an AI to a data warehouse, it sees this:

PRODGRP_NO	VARCHAR(9)
HCP_ID	NUMBER
EQ_TRX	NUMBER(38,6)

Three columns. Three cryptic names. No business meaning. The AI has no idea what these columns are, what they measure, how they relate to other tables, or what assumptions apply to their values.

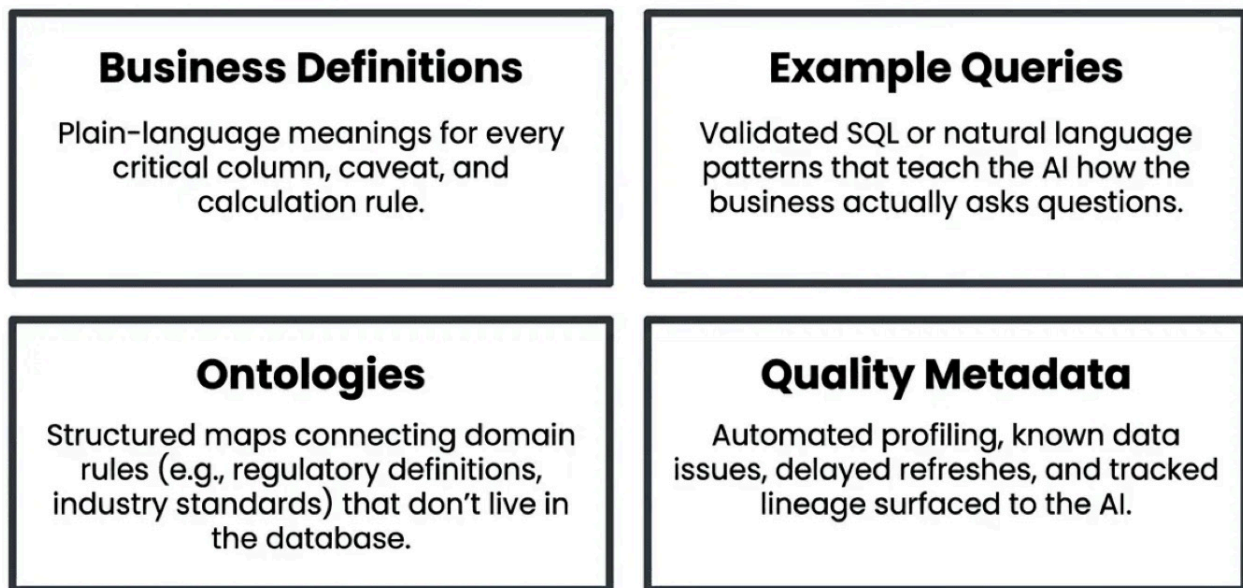
Without context, the AI is making educated guesses dressed up as answers.

Apply Context to the same data. Completely different capability. The AI now knows what it's looking at, how to join it correctly, and what the values mean. Context turns a generic chatbot into a pharma analytics expert.

What Context Includes

Context is a broad category that covers several types of information:

The Elements of Context Engineering



Every column that matters should have a plain-language definition, the business question it answers, and any caveats about how it is calculated or when it is valid. This can live in a data catalog, a README, or a structured metadata layer.

One of the most effective ways to provide context is through example queries, SQL or natural-language queries that have been validated and annotated with the correct answer. These help teach the AI how your business asks questions and what proper answers look like.

“What were total equalized prescriptions for product group X in Q3?” → annotated SQL with correct join path.

“Which territories exceeded their prescription targets last month?” → annotated SQL with correct grain and filter logic.

In industries like pharma, finance, or healthcare, there are layers of domain knowledge that aren’t captured in the data itself: regulatory definitions, industry standards, and business rules that arise from how the market operates rather than how the database is structured.

Providing the AI with an ontology , a structured map of how business concepts connect , greatly enhances the accuracy of answers to domain-specific questions.

Context also includes what you know about data quality. If a specific field has known data issues on certain dates, that counts as context. If a table’s refresh is delayed on Mondays, that is the context. If a column is deprecated and shouldn’t be used for new analysis, that is the context. DataOps Observability automatically surfaces this kind of metadata , tracking pipeline health, data freshness, test results, and lineage , and can feed it into AI context layers so the AI understands what it can and cannot rely on.

Automated schema profiling captures the statistical shape of your data: value distributions, null rates, cardinality, min/max ranges, and update frequency. Lineage traces the source of data and the transformations it has undergone. Both are essential contexts for an AI that is reasoning about data reliability and joins.

Putting It Together: The 10x Effect

Individually, DT, DX, and CTX each enhance AI-powered data analysis. Together, they create a multiplying effect. A team with Data Trust knows its AI answers are based on trustworthy data. A team with strong Data Experience ensures the AI is working with the correct tables and columns. A team that has invested in Context understands the business well enough for the AI to ask the right questions. The result isn’t just slightly better , it’s fundamentally different. Analysts spend less time validating AI outputs and more time generating insights. Business users feel more confident trusting AI answers and start developing workflows that depend on them.

The compounding effect of data engineering

$$DT \times DX \times CTX = 10x$$

Without DT

The AI confidently lies based on broken pipelines.

Without DX

The AI gets lost in a 150-table maze and times out.

Without CTX

The AI lacks business logic and outputs technically correct but practically useless answers.

With All Three

Analysts stop debugging AI outputs and start generating insights. Business users trust the answers.

A Note for Data Engineering Teams

The work described here, building data quality tests, curating semantic layers, writing business definitions, maintaining ontologies, is data engineering work. It is not glamorous. It does not ship in a sprint. But it is the infrastructure on which AI-powered analytics is built.

Every hour invested in DT, DX, and CTX reduces the hours spent debugging AI outputs by an order of magnitude. The equation is as much about return on investment as it is about architecture.

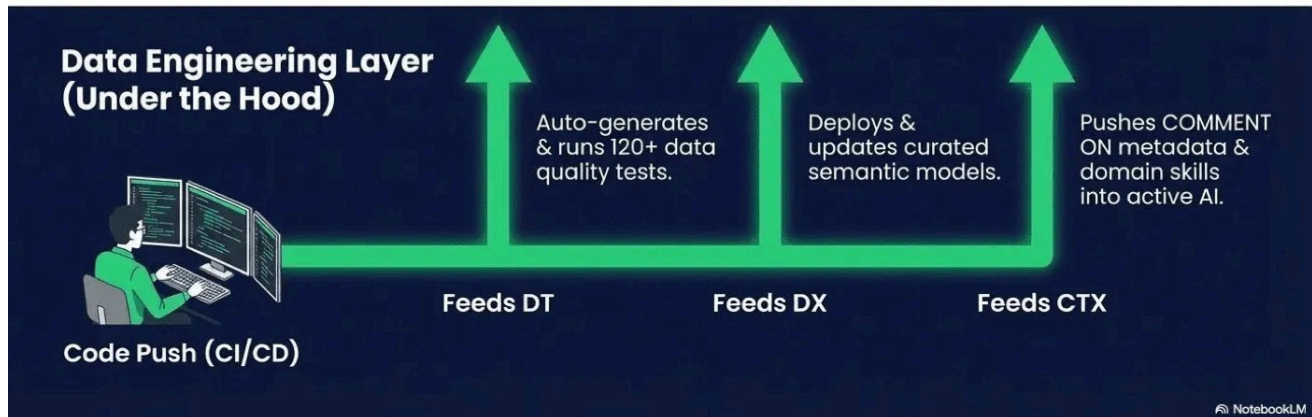
Business User & Analyst Layer (The Magic)



What is the prescriber drop in Territory 42B?

3 HCPs moved to territory 42C. Volume follows the HCPs — not a data error.

The DT + DX + CTX Translation Contract



Getting Started

You don't have to solve all three dimensions at once. Start where the pain is greatest:

Start with DT: If your team doesn't trust the data, start with DT. Run automated quality tests on your most critical tables. Use TestGen or a similar tool to profile your schemas and surface issues.

Start with DX: If analysts are wasting time navigating a confusing data landscape, begin there. Identify the ten tables that answer 80% of your business questions and build a clean semantic layer over them.

Start with CTX: If your AI gives technically correct but business-wise wrong answers, start with CTX. Write business definitions for your most important columns. Add five example queries. Build a data dictionary for the domain in which your AI operates.

The equation does not require perfection in any dimension. It requires meaningful progress in all three. A little more trust, a cleaner experience, and richer context compound into dramatically better AI-powered analysis.

How to diagnose and start fixing your data foundation

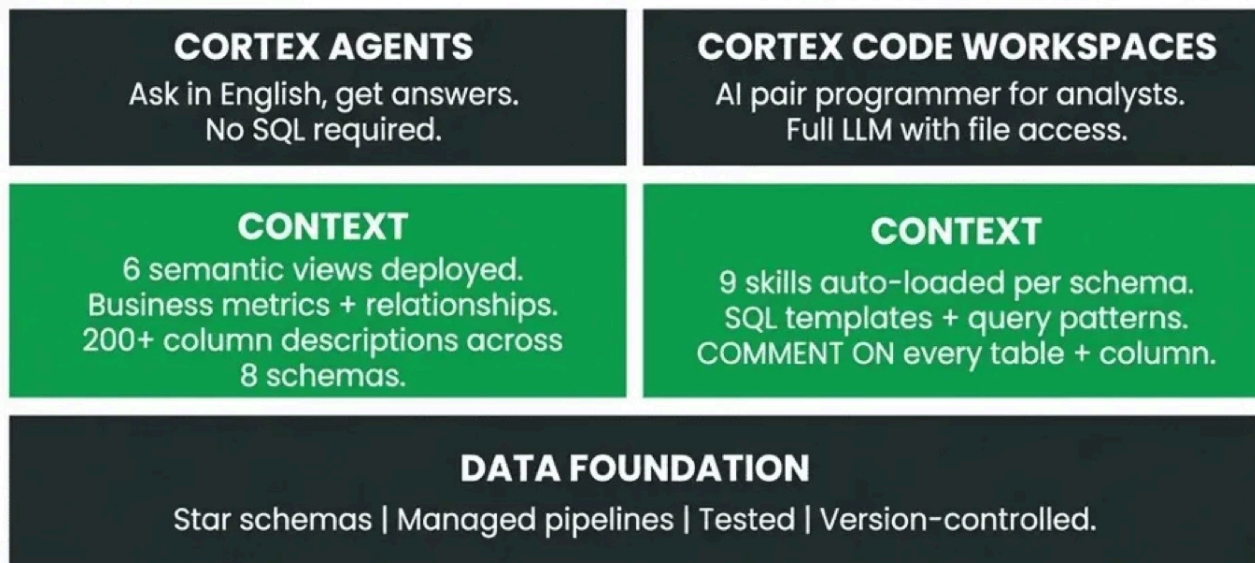
DT	Symptom: Your team doesn't trust the AI's answers.	Diagnosis -> First Step: Start with DT. Run automated quality tests on your top critical tables to establish a baseline.
DX	Symptom: Analysts waste hours finding the right data.	Diagnosis -> First Step: Start with DX. Identify the 10 tables that answer 80% of questions and build a clean semantic front door.
CTX	Symptom: AI gives technically correct but business-wrong answers.	Diagnosis -> First Step: Start with CTX. Write clear business definitions for top columns and add 5 verified example queries.

Real-World Example: Commercial Pharma Analytics on Snowflake & Databricks

The DT + DX + CTX framework is not abstract. Here is what it looks like deployed against a real pharma data environment running on Snowflake, with IQVIA data, field sales teams, and analysts who need answers without writing SQL.

The Architecture: Two Paths, One Foundation

When context engineering is done properly, it enables two distinct AI-powered analytics modes to run simultaneously , both on the same underlying data foundation.



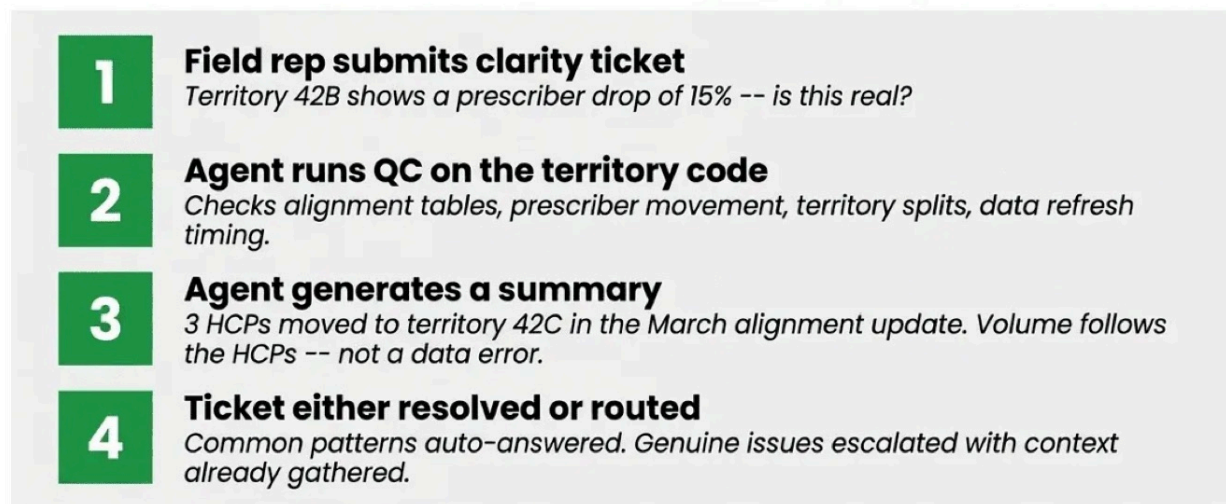
Both paths run on the same data foundation , star schemas, managed pipelines, tested and version-controlled. Context is what makes both accurate. Both paths are running simultaneously on production data.

Example: The Clarity Ticket Agent

With the right data foundation, context, and AI tooling in place, entirely new workflows become possible. Here is one concrete use case: an agent that automatically handles tickets for field data discrepancies.

AI in Action: The Clarity Ticket Agent

Deflecting field data discrepancy tickets using context-aware AI.



Possible Phase 1 target: pilot focused on clarity tickets as first agent use case.

Field reps routinely submit tickets when the data does not match their expectations , a territory showing a sudden drop in prescribers, a product group with anomalous volume. This report does not reconcile with what they see in the field. Today, these tickets are routed to analysts. With a context-aware AI agent, many can be resolved instantly:

This agent is possible only because the data underneath it is trusted (DT), the schema is navigable (DX), and the context layer understands what territory alignment tables mean, how HCP movement is recorded, and what data refresh timing looks like (CTX). Without all three, the agent would hallucinate or confidently give wrong answers.

What the Team Actually Gets

When DT + DX + CTX is in place, the benefits are not uniform across the organization , they are specific to each type of user:

What Your Team Actually Gets

Business Users

Sales Ops, Market Access, Brand Analytics, Managed Care

- Ask questions in plain English
- Get answers in seconds
- No SQL required
- Self-serve via Snowsight browser

Analysts & Data Teams

- AI-powered workspaces with Cortex Code that knows your schemas
- Ready-to-run query library
- Documented business definitions in the platform
- Fewer "can you run this for me?" interruptions

Leadership

- Institutional knowledge captured in the platform
- Faster time-to-insight for market questions
- Foundation for advanced analytics
- AI adoption that meets users where they are

Context Engineering Is Platform-Portable

One important property of the CTX investment is its durability. The context you build , business definitions, semantic views, query patterns, domain skills , maps directly across platforms. If your organization moves from Snowflake to Databricks, or runs both, the content travels with you.

What We Build	Snowflake	Databricks
Natural language agents	Cortex Agents	AI/BI Genie
Semantic definitions	Semantic Views	Unity Catalog Metric Views
Verified/trusted queries	Verified Queries	Trusted Assets
AI coding assistant	Cortex Code	Databricks Assistant
Domain skills	Skills (auto-loaded)	Custom Skills (auto-loaded)
Feedback & observability	AI Observability Events	MLflow Tracing

Your investment in context engineering is durable. The platform is just where it runs.

Conclusion

The promise of AI-powered data analysis is real. Natural language queries, automated insight generation, and super-powered data analysis , these capabilities are available today. But they only deliver their full value when the data underneath them is trustworthy, legible, and enriched with context.

$$DT + DX + CTX = 10x$$

The teams that win with AI are not the ones that deploy the most sophisticated models.

They are the ones who do the foundational data work that makes any model effective.

DataKitchen builds open-source software for data quality testing and DataOps observability. TestGen is free at datakitchen.io.



DT + DX + CTX = 10x is not a slogan. It is a checklist for the infrastructure work that makes AI analysis reliable, accurate, and genuinely useful to the people who depend on it.

The teams that will win with AI are not the ones that deploy the most sophisticated models. They are the ones who do the foundational data work that makes any model effective.

An Addendum for Data Engineering

The 10x opportunity from LLMs is not limited to analysts. Data engineers building pipelines face the same foundational challenge: AI is an amplifier, and if the data flowing through those pipelines is untrustworthy, poorly tested, or missing operational context, it can amplify bad outputs at scale. We have a companion framework for that challenge in The Equation for AI Success: **Claude Code (DataOps + FITT + Data Testing) = 10x Data Engineering**. Where this post addresses how analysts make their data legible, trusted, and context-rich enough for AI to reason over it reliably, that one addresses how data engineers build the pipelines that make AI safe and productive. The two equations are meant to work together.

Opportunity to use AI to 10x the productivity of:

The Integration Ecosystem
Data Engineers who are integrating the data

DataOps + FITT + Data Testing = 10x

The Insight Ecosystem
Data Users directly using the data to give business insight

DT + DX + CTX = 10x

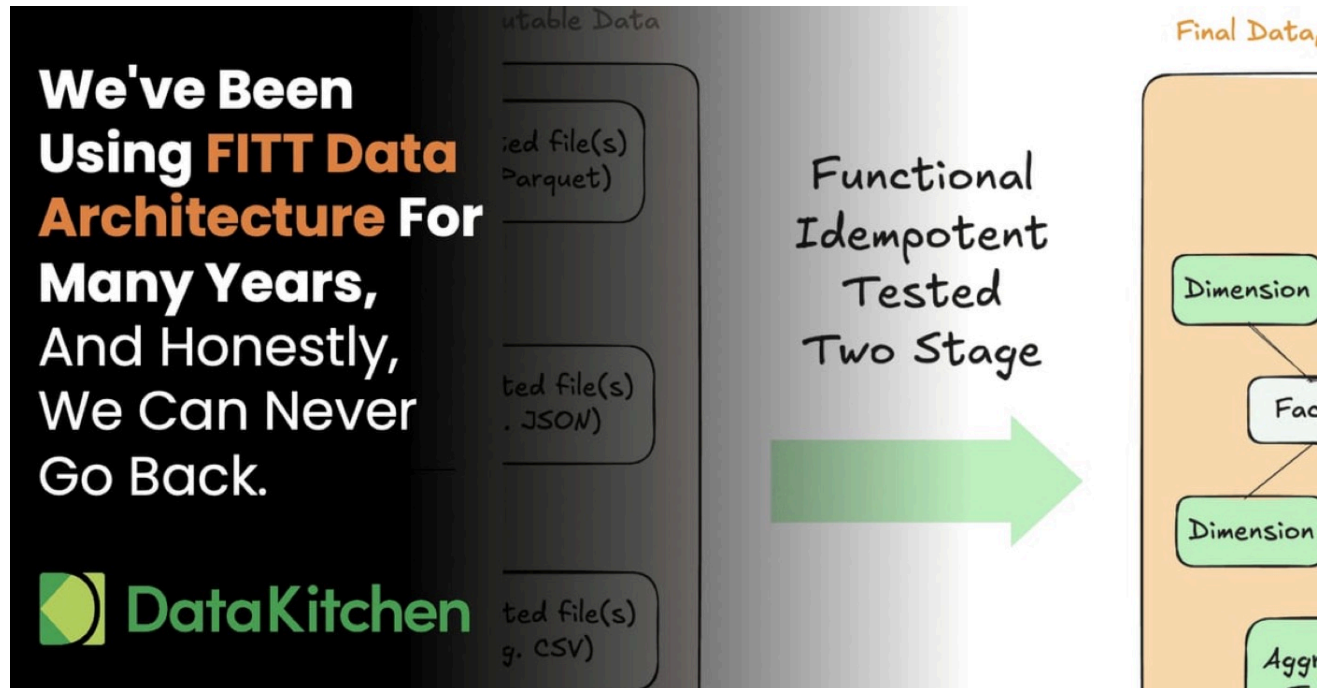
The architectural foundation is what makes the acceleration safe across the entire enterprise.

DataKitchen

Read the companion piece: [The Equation for AI Success: Claude Code + DataOps + FITT + Data Testing = 10x](#)

We've Been Using FITT Data Architecture For Many Years, And Honestly, We Can Never Go Back

Read the original article at: <https://datakitchen.io/blog/fitt-data-architecture/>



Picture this: It's 2:47 AM, your Slack is buzzing with alerts, and the CFO's quarterly report is broken because somewhere in your seven-layer medallion architecture, the bronze data doesn't match the gold data. Sound familiar? We lived this nightmare for years until we discovered something that changed everything about how we approach data engineering.

We wanted to make our lives easier, simpler, and less complicated. The cloud has made it incredibly affordable to have copies of systems, tools, pipelines, and even data. We craved a single source of truth through Git and grew tired of managing inconsistent data across environments. Most importantly, we wanted zero meaningful difference between development, production, and canary environments. The goal was simple: complete a rebuild, push a minor update, add a new dataset, or recreate last month's results without breaking a sweat.

IMPORTANT

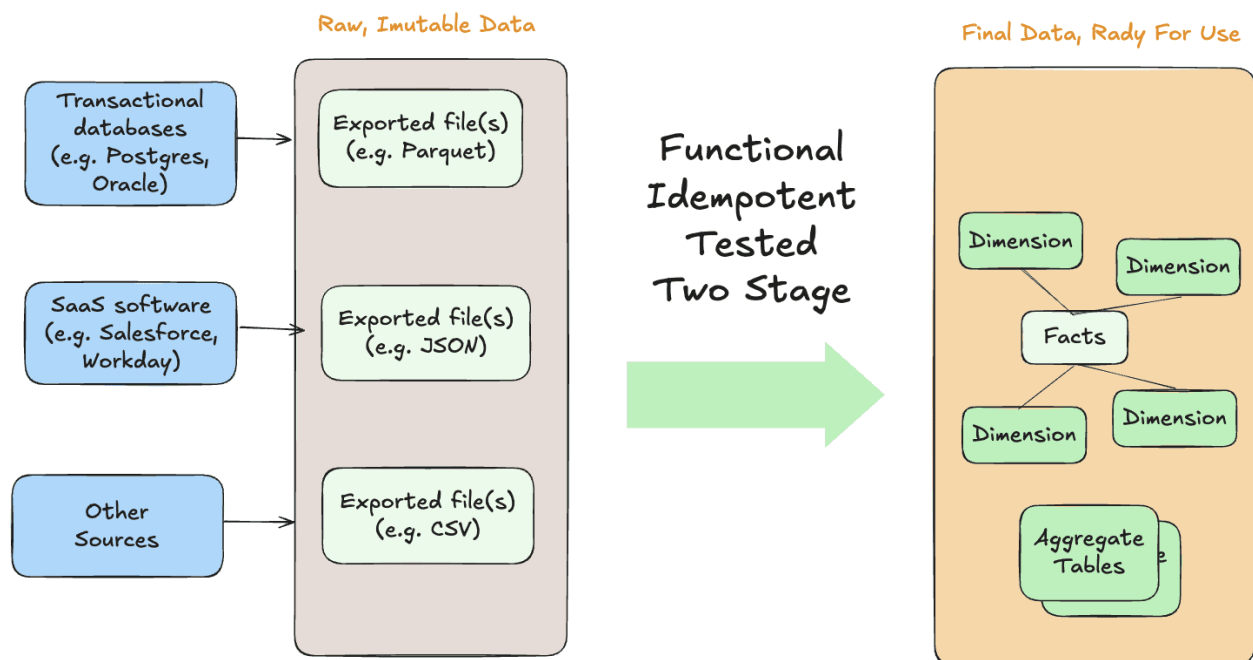
Data architecture should focus on maximizing data engineer productivity and simplicity while reducing complexity

The alternative, maintaining three to five copies of data in every environment and spending entire weekends debugging why Level 1 data differs from Level 3 data, is unsustainable. If explaining a minor fix to a junior data engineer requires a PhD in your data architecture, then your architecture has become the problem.

We've been applying these principles in our data engineering consulting for many years, and it's time we gave them a proper name and explained why they've been transformative for every team we've worked with. Yes, I know, another acronym in a field already drowning in them. But this isn't just buzzword engineering; this represents how we learned to stop breaking things constantly and started working faster with less risk, fear, and late-night heroics.

What is FITT Data Architecture?

FITT represents four principles we follow religiously, born from countless production incidents and hard-learned lessons about what works in real-world data systems.



Functional - Everything is Pure Functions

No more mystery dependencies or hidden state lurking in your transformations. For every version of code, each transformation takes clearly defined inputs and produces predictable outputs, period. Same input equals same output, every single time, regardless of when or where you run it.

We have decided to treat all raw data as immutable by default. Yes, storage and compute have costs, but debugging at 2 AM is infinitely more expensive. The math is simple: data engineering time is worth more than compute costs, which are worth more than storage costs. When you embrace this hierarchy, many architectural decisions become obvious.

IMPORTANT

For every version of code, each transformation takes input and produces output, period.

This functional approach eliminates an entire class of bugs that plague traditional data systems. No more wondering if yesterday's transformation had any impact on today's results. No more state-dependent edge cases that only surface in production under specific conditions. Each transformation becomes a mathematical function that you can reason about, test, and trust.

Consider a typical calculation of customer lifetime value. In a traditional system, this might depend on cached aggregates or previous run results. In a functional system, the calculation receives raw transaction data and customer attributes as input and produces CLV metrics as output. Run it in January with December's data, and you'll get identical results to running it in December, guaranteed.

Idempotent - Run it 1000 Times, Get the Same Result

This principle fundamentally changed how our teams approach data engineering. When everything is idempotent, recovery becomes trivial and stress levels plummet. Pipeline broke due to a schema change? Simply fix the transformation and re-run it. Getting weird results that don't match expectations? Re-run the pipeline with debugging enabled. Want to test a change safely? Re-run it on yesterday's data and compare outputs.

The psychological impact of this approach cannot be overstated. Our junior developers are now allowed to experiment, rather than living in constant fear of breaking production systems. They've learned that curiosity and iteration are safe when the underlying system is designed to handle repetition gracefully.

IMPORTANT

This one changed our lives. When everything is idempotent, recovery becomes trivial. Pipeline broke? Just re-run it. Weird results? Re-run it. Want to test a change? Re-run it on yesterday's data. Need to backfill from 80% done. Just run the substeps. The psychological impact is enormous

Idempotency proves crucial for several scenarios that every data team faces regularly. During recovery situations, when something inevitably breaks, you can simply modify the code or parameters and re-run the affected portions rather than spending hours debugging. For consistency, development and production environments behave identically because they're running the same idempotent processes. Most importantly, engineers gain confidence to experiment and iterate without the paralyzing fear of corrupting critical business data.

This property extends beyond individual transformations to entire pipeline orchestration. Need to backfill three months of data after fixing a bug? The process is identical whether you're filling one day or one hundred days. Each day's processing is self-contained and doesn't depend on the success or failure of adjacent days.

The idempotent approach naturally encourages a "build a little, test a little, learn a lot" development rhythm. Consider implementing a complex customer segmentation model that involves multiple data sources, feature engineering, and machine learning predictions. Rather than building the entire pipeline and hoping it works, you can break it into small, idempotent chunks: first, transform raw customer data into clean features and test against known customers; then add transaction aggregations and verify the

math with sample accounts; finally, integrate the ML predictions and validate outputs against business expectations. Each step can be developed, tested, and refined independently using the same historical data. When a transformation doesn't produce expected results, you haven't lost hours of downstream processing, you simply adjust the logic and re-run that specific chunk. This granular approach significantly accelerates development cycles, as engineers can iterate on individual components without having to rebuild entire pipelines.

Tested - Automated Tests Everywhere, Hope is Not a Strategy

We treat tests as first-class citizens in our data architecture, not afterthoughts or nice-to-haves. Every pipeline includes comprehensive data quality tests covering schema validation, null checks, and range verification. Business logic tests ensure transformation correctness by validating that known inputs produce expected outputs. Pipeline behavior tests verify that the system handles edge cases, failures, and recovery scenarios in a graceful manner. SLA checks monitor performance and alert when processing times exceed acceptable thresholds.

These tests aren't just quality assurance mechanisms, they serve as living documentation of what the system is intended to accomplish. When a new engineer joins the team, they can read the tests to understand both the expected behavior and the edge cases that the system handles. Six years ago, one of our VP customers boasted of having run 10,000 tests against his production data - so, when someone questioned the data, he could say, "Our 10,000 tests didn't catch it. Are you sure?"

IMPORTANT

We treat tests like first-class citizens. Tests aren't just QA - they're living proof of what the system should do. And by 'tests,' we mean numerous tests, hundreds, thousands, covering every table.

The key insight is that data systems fail in predictable ways, and comprehensive testing catches these failures early, when they're cheap to fix, rather than late, when they become expensive disasters. A missing null check was detected in the development costs, which can be fixed in minutes. The same issue discovered in production during a board presentation costs significantly more in both engineering time and business credibility.

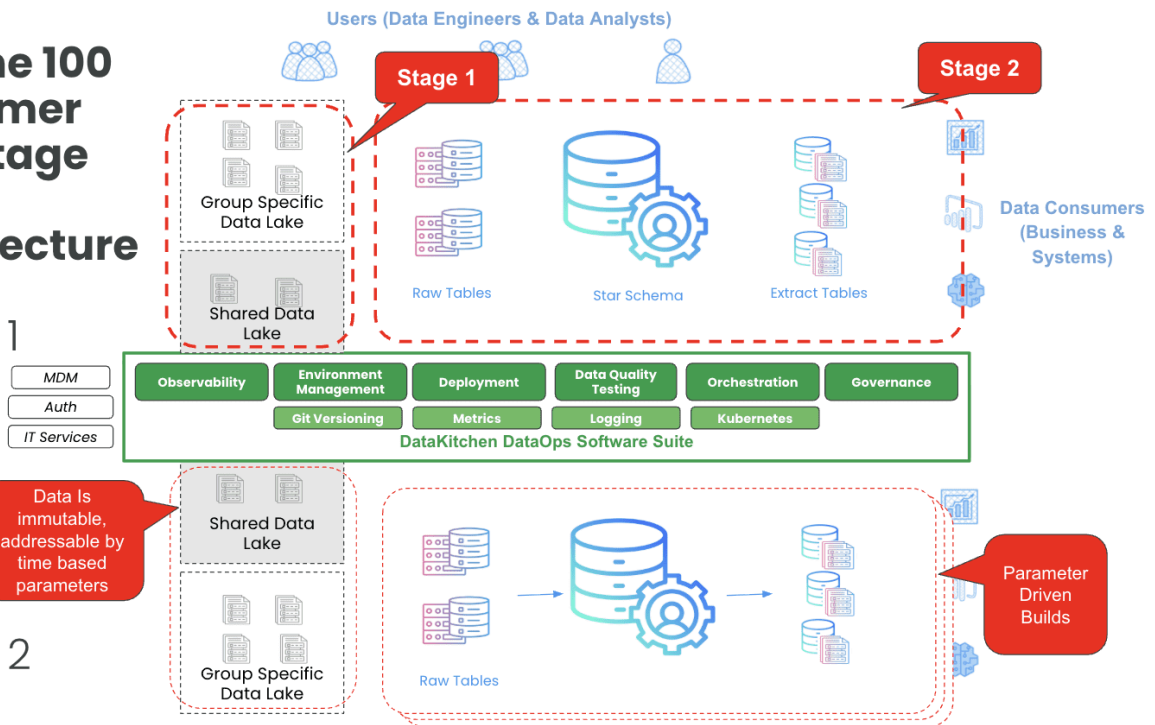
Modern data testing frameworks enable the maintenance of thousands of tests across hundreds of tables. These tests run automatically with every deployment and provide immediate feedback when changes introduce regressions. The investment in test infrastructure pays dividends in reduced on-call burden and increased confidence in system changes.

Two-stage - Raw to Final, That's It

This principle represents our most controversial departure from conventional wisdom, and it's also been our most liberating change. We completely discarded our multi stage architecture and adopted a radically simplified two-stage approach.

Fortune 100 Customer Two Stage Data Architecture

Group 1



Group 2

The first stage, raw data, represents exactly what we received from source systems, immutable, never modified, and always preserving the original context and timestamp of ingestion. The second stage, final data, contains information ready for consumption by downstream systems, analytics tools, and business users.

IMPORTANT

Raw, which represents exactly what we received from source systems, immutable forever. Final, ready for consumption. Everything in between? Can be deleted

Everything between these stages? Ephemeral, temporary, and disposable. These intermediate transformations can be deleted after processing or kept around for debugging, the choice depends on operational needs rather than architectural requirements.

“But what about bronze, silver, and gold layers?” you might ask. Through years of experience, we realized those layers often represented arbitrary complexity rather than meaningful business value. The benefits traditionally achieved through staged processing, data quality, transformation logic, and performance optimization, are now accomplished through functional composition and comprehensive testing.

This approach creates what we refer to as a “copy-paste” data architecture. Do you want an exact copy of the production for testing? Simply adjust a few configuration parameters, create a Git branch, and rebuild the environment. The raw data provides your foundation, and the functional transformations recreate everything else deterministically.

Development vs Production: A Pattern That Works

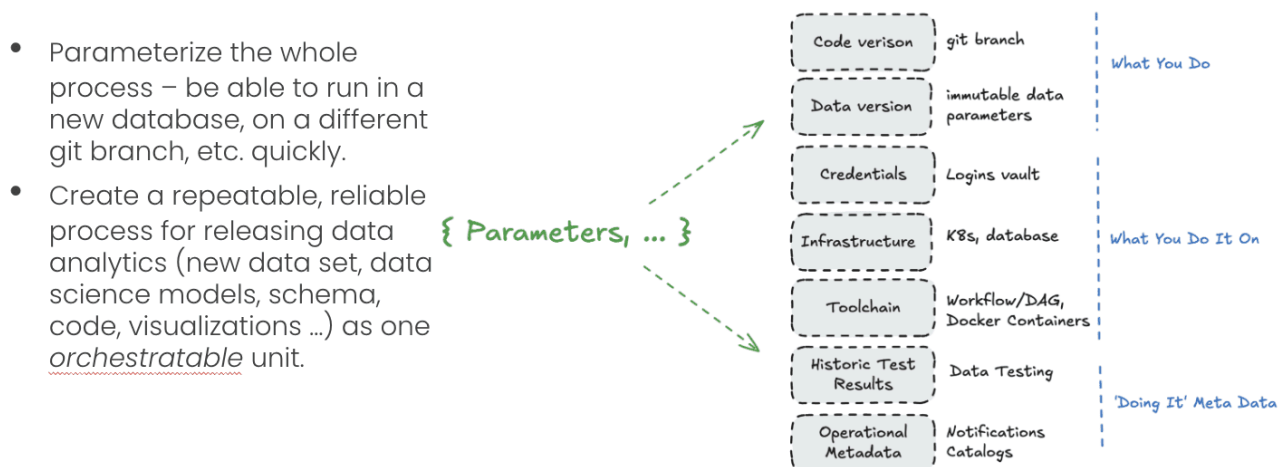
The implementation of FITT principles requires rethinking how we manage environments and deployments. Our approach employs a simple yet powerful pattern: development environments run yesterday's data with today's code, while production environments run today's data with yesterday's code.

IMPORTANT

Dev: Yesterday's data, today's code (we can have multiple environments), Prod: Today's data, yesterday's code (single environment, stable)

This pattern ensures we never deploy untested code to production while allowing developers to iterate safely without affecting real business data. Version control becomes the single source of truth for both code and configuration. Our Git repository essentially IS our pipeline definition.

Want to reproduce results from six months ago? Pull from git and re-run the pipeline against the archived raw data. The combination of immutable raw data and version-controlled transformations makes time travel trivial. This capability proves invaluable during audit seasons, regulatory reviews, or when investigating anomalies in historical data.



The development pattern enables proper continuous integration for data pipelines. Each pull request can be validated against historical data, ensuring that proposed changes produce expected results across various scenarios. Code reviews become more meaningful because reviewers can test changes rather than just inspecting code.

Incremental updates still work within this framework, the crucial insight is making each increment idempotent. Whether you process today's incremental update once or fifty times, you achieve identical results. This property eliminates an entire class of timing-related bugs that plague traditional incremental processing systems.

The Awesome Results

After implementing FITT principles across dozens of client engagements, the results have been consistently remarkable. Incidents have dropped so dramatically that customers frequently comment that our systems are “too quiet”, they’ve grown accustomed to constant alerts from their previous data infrastructure.

When issues do arise, recovery follows a predictable pattern: identify the problematic component, fix the transformation logic, and re-run the affected pipeline segments. No more archaeological expeditions or complex rollback procedures that might introduce additional problems.

New team members achieve productivity within days rather than months because the system behaves predictably and follows consistent patterns. Junior engineers can confidently make changes because the comprehensive testing catches mistakes early and the idempotent nature ensures that failures don’t create lasting damage.

IMPORTANT

If you have to explain how to make a minor fix to a junior data engineer, your data architecture is the problem.

Data quality problems surface during development rather than production, thanks to comprehensive testing integrated into the development workflow. Mysterious data drift, that phenomenon where numbers slowly change over time for unknown reasons, becomes impossible because every change in output data can be traced to a specific Git commit.

Perhaps most importantly, the architecture scales with team growth. Adding new data sources, implementing additional transformations, or expanding to new use cases follows the same patterns that existing team members already understand. Knowledge transfer occurs through code and tests, rather than relying on tribal knowledge and lengthy documentation.

Addressing Common Objections

The most frequent pushback we encounter centers around cost concerns. “Storage costs will kill us!” teams often protest. However, we’re specifically talking about raw data storage, not maintaining every intermediate table indefinitely. Modern cloud storage costs have plummeted to the point where the storage expense of raw data is negligible compared to engineering time spent debugging data quality issues. FITT principles also help reduce computing costs. When pipelines are idempotent and well-tested, you eliminate the expensive computational overhead of debugging failed runs, reprocessing corrupted data, and running complex recovery procedures that often consume more resources than the original processing.

IMPORTANT

Data Engineering time >> compute costs > storage costs.

Performance concerns represent another common objection. The reality is that when engineers aren't constantly debugging broken states and hunting down data inconsistencies, they have significantly more time to optimize the systems that impact user experience. Clean, functional architectures are inherently easier to optimize because performance bottlenecks become obvious rather than hidden behind complex interdependencies.

Getting Started with FITT Implementation

Implementing FITT doesn't require discarding existing infrastructure or rewriting every pipeline simultaneously. The transition can happen incrementally, allowing teams to gain confidence with the approach before committing fully.

- Start by selecting one critical pipeline, ideally something that causes regular pain but isn't so complex that failure would be catastrophic. Refactor this pipeline according to the FITT principles, treating it as a proof of concept for the broader architectural approach.
- Begin treating raw data as truly immutable, even if your current systems don't enforce this property. This mental shift alone eliminates many classes of debugging scenarios and forces clearer thinking about transformation logic.
- Add comprehensive data tests to existing transformations before modifying them. This safety net catches regressions during refactoring and provides confidence that changes maintain expected behavior.

IMPORTANT

Start saying 'yes' to user requests , I can get that done tomorrow

- Simplify staging by identifying and eliminating intermediate layers that don't provide clear value. Many organizations discover that their bronze and silver layers exist primarily for historical reasons rather than current business requirements.
- Ensure that all transformation logic and data testing code are stored in version control with proper branching strategies. This step enables the development versus production patterns that make safe iteration possible.

Be Data Engineering Productivity-Centric, Not Data Architecture-Centric

Traditional data architectures focus on organizing data, determining where information lives, how it's structured, and how it flows between systems. FITT represents a philosophical shift toward centering architecture around team processes and productivity. This isn't merely academic; it's a practical response to the reality that data systems fail not because of storage limitations, but because of unreliable, untestable, and irreproducible transformations.

The human element represents the largest expense on most data teams. Optimizing for engineer productivity and satisfaction creates a virtuous cycle where better systems attract top talent, which in turn builds even better systems.

IMPORTANT

Medallion data architectures are designed to maximize data vendor revenues, not data team productivity.

The core insight driving FITT is elegantly simple: data represents the byproduct of functions, not the primary design consideration. When we design around processes and transformations, we create systems that are inherently more reliable, debuggable, and maintainable.

Teams implementing FITT principles consistently report significant improvements across multiple dimensions. Developer productivity increases as engineers spend less time debugging mysterious failures and more time building valuable features. System reliability improves through comprehensive testing and idempotent operations. Data quality becomes proactive rather than reactive through testing integrated into development workflows. Operational complexity decreases through fewer moving parts and clearer failure modes.

The Honest Downsides

Transparency requires acknowledging that FITT implementation comes with genuine trade-offs. Initial setup takes longer than traditional approaches because comprehensive testing and functional design require upfront investment. The benefits compound over time, but teams must be prepared for a higher initial setup cost.

Some team members resist the constraints, particularly engineers accustomed to imperative programming styles or those who view testing as optional overhead. Change management becomes crucial for successful adoption, requiring clear communication about the long-term benefits that come from short-term discipline.

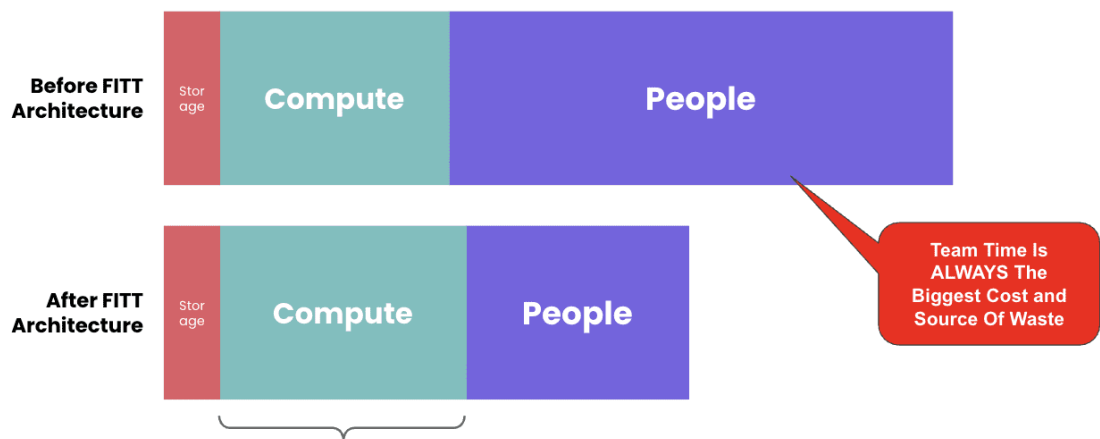
IMPORTANT

You need to have tests, lots of them. But tests are the gift you give to your future self.

Better tooling becomes necessary for writing and maintaining comprehensive data quality tests. Teams need access to modern data testing frameworks, such as [DataKitchen's TestGen](#), or must write a large number of [tests to achieve coverage](#).

Equally important is choosing an orchestrator that integrates testing into the process flow rather than treating it as an afterthought. Modern orchestration platforms, such as Dagster, provide native data quality testing as part of their execution model, making it natural to validate data at each step of transformation. DataKitchen's [DataOps Automation](#) platform builds testing directly into pipeline orchestration, ensuring that data quality checks are mandatory rather than optional. Even traditional orchestrators, such as Airflow, can effectively support FITT principles when combined with the right plugins and patterns that enforce testing at each stage. The key is selecting tools that make comprehensive testing the default behavior rather than something engineers must remember to add.

Conclusion: Focus On Reducing Your Biggest Costs



You database and compute cost may go up somewhat. Compute cost is more than offset by reduction (or efficiency increase) from data team productivity

Why You Should Embrace This Approach

The data engineering field suffers from **endemic burnout**, **constant firefighting**, and teams spending disproportionate time on operational issues rather than building valuable solutions. FITT isn't magic, but it represents the closest approach we've discovered to making data engineering "boring" in the best possible sense, predictable, reliable, and stress-free.

The transformation from **hero-driven development**, where only senior engineers dare touch production systems, to an environment where any team member can contribute confidently represents the real victory. This democratization of system ownership scales teams more effectively than hiring additional senior talent.

Junior engineers thrive in FITT environments because the system's predictable behavior enables them to develop intuition about data engineering without fear of causing catastrophic failures. Senior engineers appreciate the reduced cognitive load, allowing them to focus on high-value coding rather than constantly firefighting.

IMPORTANT

What's important? Process acting on data >> data itself. Why? Cognitive load

The approach has proven successful across various industries, team sizes, and technical stacks. Whether you're processing financial transactions, customer behavior data, or IoT sensor readings, the fundamental principles of functional design, idempotent operations, comprehensive testing, and simplified staging apply universally.

Most importantly, FITT enables data teams to focus on their core mission: transforming raw information into business value. When the infrastructure becomes reliable and predictable, engineers can focus on understanding business requirements, optimizing for user needs, and developing innovative solutions rather than maintaining fragile systems.

IMPORTANT

Most data architectures are overly complex, trend following, and not built for rapid change, high quality, and ease of work.

The choice is ultimately simple: continue accepting the current state of data engineering chaos as inevitable, or invest in systematic approaches that make the work sustainable and enjoyable. FITT provides a proven path toward the latter, backed by years of real-world implementation across diverse environments.

Every data team deserves to experience the satisfaction of systems that just work, deployments that happen without drama, and on-call rotations that rarely interrupt personal time. FITT makes this vision achievable through the systematic application of proven principles, rather than relying on luck with the next architectural trend. We run it on commercial pharma launches, where the vendor feeds arrive late and get restated: see [FITT applied to a commercial pharma data platform](#).

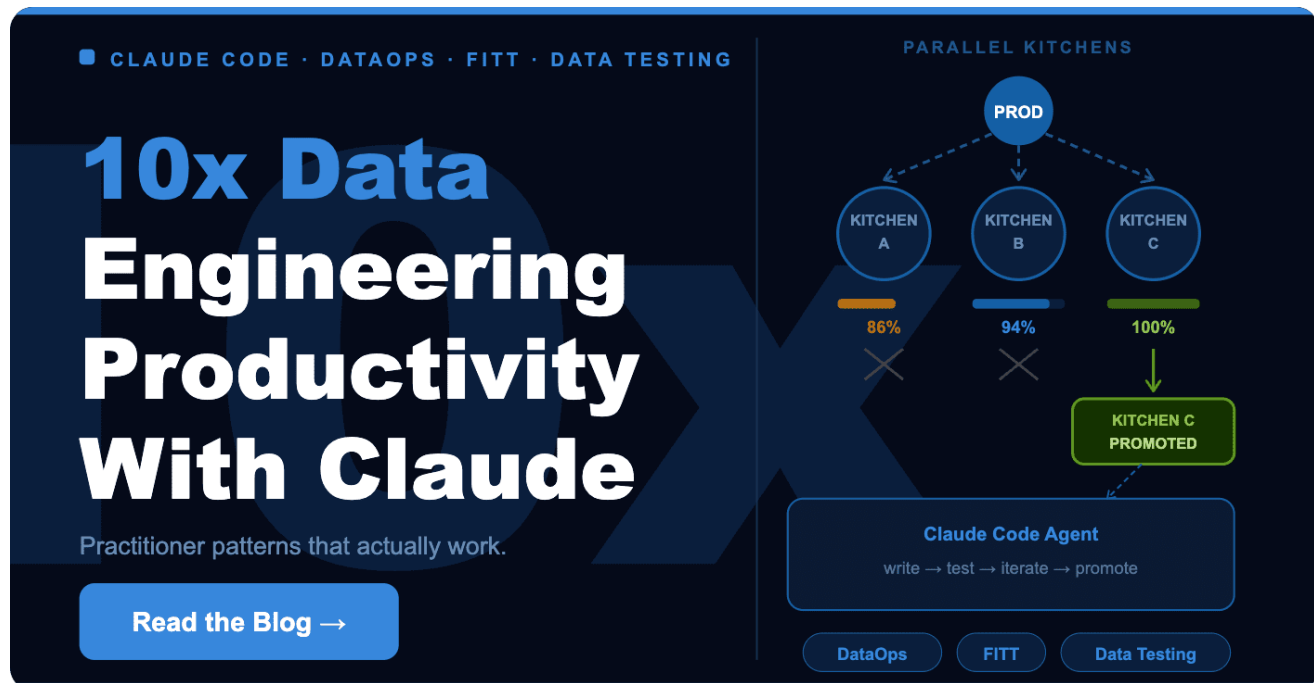
Want to learn about this in a Webinar format? [The FITT Way To Data Products: A New Data Architecture For A Product-Centric World](#)

More reading

- [Functional Data Engineering - A Blueprint](#)
- [Functional Data Engineering](#) , a modern paradigm for batch data processing
- [easy-functional-data-engineering](#)
- [Idempotent Data Pipeline](#)
- [Lars Albertsson - All the DataOps, all the paradigms](#)

DataOps + FITT + Data Testing = 10x Data Engineering Productivity with AI

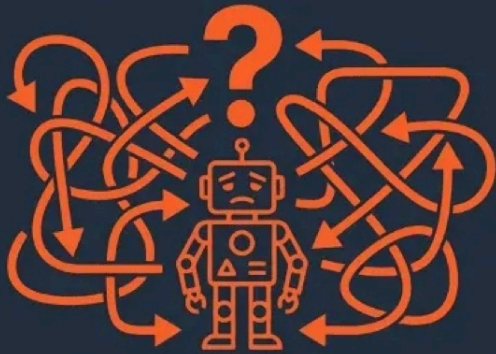
Read the original article at: <https://datakitchen.io/blog/dataops-fitt-data-testing-10x-data-engineering-productivity-with-ai/>



10x Your Data Engineering with AI: Practitioner Patterns That Actually Work

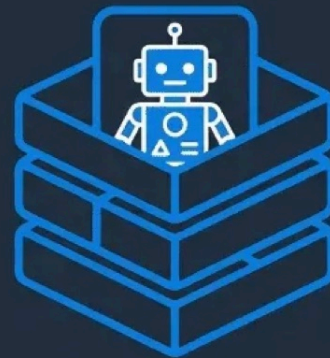
AI coding tools like Claude Code are generating significant excitement in software engineering. But for data engineers, getting 10 times the productivity isn't automatic. Just adding an AI agent to a messy pipeline and hoping it works usually leads to frustration.

AI is not a magic data tool. It is a coding and process engine.



The Myth.

Dropping an AI agent into an unstructured codebase without isolated testing means flying blind.



The Reality

The 10x productivity leap requires treating data engineering as a process problem, building a foundational architecture where agents can safely iterate.

At DataKitchen, we use Claude Code every day in our data engineering consulting work. We've found that the productivity boost is real, but it means changing how you approach your work. Most data teams focus on data itself: tables, metadata, schemas, and datasets. Claude Code isn't a data tool; it's about coding and processes acting upon data. To get the most out of it, you need to think like a **process-centric data team**, organizing work around pipelines, workflows, and transforms instead of just tables. Once you do that, the right foundation becomes clear: DataOps principles, FITT data architecture, and solid data testing all working together.

We summed up what makes this combination effective in a simple equation:

Claude Code(DataOps + FITT + Data Testing) = 10x Data Engineering Productivity

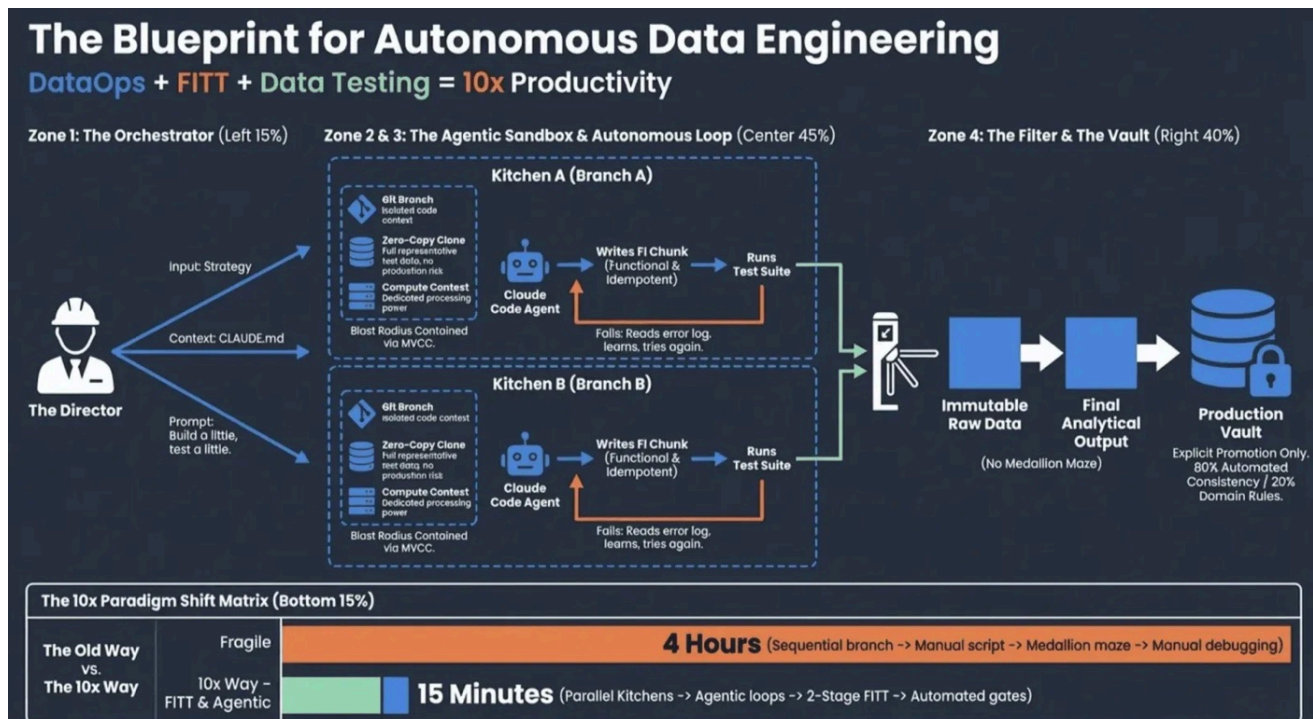
In this post, we'll break down each part and explain why all three are needed to unlock AI-assisted data engineering's full potential.

Why AI Alone Is Not Enough

Claude Code is genuinely impressive. Given a clear task, it can write SQL transforms, build ingestion logic, debug schema mismatches, and keep iterating on solutions for 20 minutes or more without human input. But capability alone is not enough. An AI agent working in a messy codebase, without isolated environments or tests to validate its output, has no reliable way to know whether what it produced is correct.

Understanding why requires understanding how Claude Code actually works. It does not write code once and hand it over. It operates in an agentic loop: writing a solution, running it, reading the output, diagnosing what broke, and trying again. That cycle repeats until the output satisfies the goal. The

iterations happen quickly, and the intermediate failures are invisible to the user, so the result can feel instantaneous. But what is actually happening is disciplined, self-directed trial and error, closer to how a skilled developer thinks than to how a code autocomplete tool operates.



That loop only works when the environment supports it. If Claude is iterating against a shared production database, every failed attempt is a risk. If there are no tests, Claude has no signal for when it is done or whether it has regressed something upstream. The agentic loop and the infrastructure underneath it are not separate concerns. One depends entirely on the other.

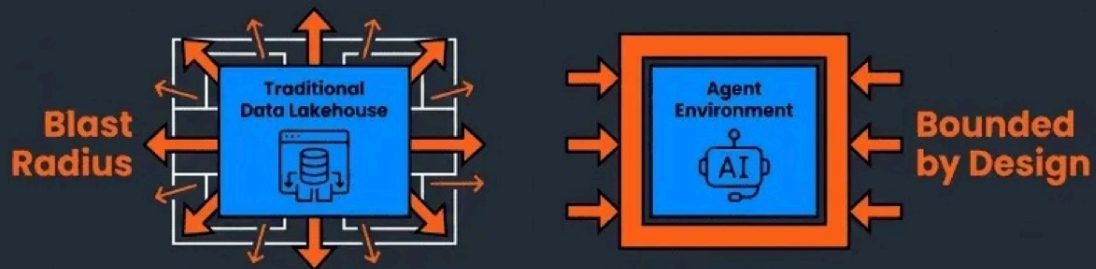
The data engineers getting 10x results have invested in the scaffolding that makes the loop safe and productive. That scaffolding has three layers

Layer 1: DataOps

The Trust Problem

Even as AI capabilities improve rapidly, most enterprises are not ready to let agents loose on production data. The fears are concrete and legitimate: an agent accidentally dropping a table, polluting the data lake with hallucinated outputs, or silently corrupting a dimension that feeds downstream reporting. These are not hypothetical failure modes. They are the natural result of handing a powerful autonomous tool access to infrastructure that was never designed for agent access patterns.

The Trust Problem: Why lakehouses break down under agent workloads



		Traditional Lakehouse	Agent-Ready Infrastructure
1	Design Paradigm	Human-operated, shared environments	API-based access, concurrent branching
2	Safety Mechanism	Human-readable audit trails, manual rollbacks	Ephemeral isolated environments, bounded by design
3	Failure Mode	Agents drop production tables or hallucinate data	Blast radius contained to isolated sandbox

This trust gap is the real blocker for agentic data engineering adoption. It is not a capability problem. Claude Code can do the work. It is an infrastructure problem. Traditional lakehouses were built for human-operated workflows with human-readable audit trails and human-managed rollback procedures. AI Agents like Claude Code need something different.

The path to trustworthy agentic workflows starts with solving the infrastructure problem first. Design the data environment around agent access patterns, and governance follows naturally. Try to bolt governance onto an environment that was not designed for agents, and you are constantly fighting the architecture.

The Solution: Isolated Environments

DataOps is the application of Agile and DevOps principles to data work. The [DataOps Cookbook](#) is the definitive guide to these principles, and for AI-assisted data engineering, the most critical concept it surfaces is environment branching, not just code branching. Claude Code works through trial and error. Given a task, it will attempt a solution, run it, evaluate the result, and try again. This iteration loop can run for 30 minutes, an hour, or longer. During that time, you do not want it touching your production environment or your main branch.

The key insight is that correct concurrent AI workloads require solving the data, compute, and code branch isolation through a unified API. This is the same problem that database systems solved decades ago with MVCC (Multi-Version Concurrency Control). MVCC is the mechanism that gives each database user the illusion that they are the only user of the system. Every transaction sees a consistent snapshot of the data. Changes are isolated until committed. Other users are never exposed to work in progress.

Layer 1: DataOps and the 'Kitchen' Isolation Model

Concurrent AI workloads require **Multi-Version Concurrency Control (MVCC)** for pipelines.

Production Branch



Claude enters the Kitchen, runs trial-and-error loops unattended, and either fails safely or produces a clean dish. Production remains untouched until explicit promotion.

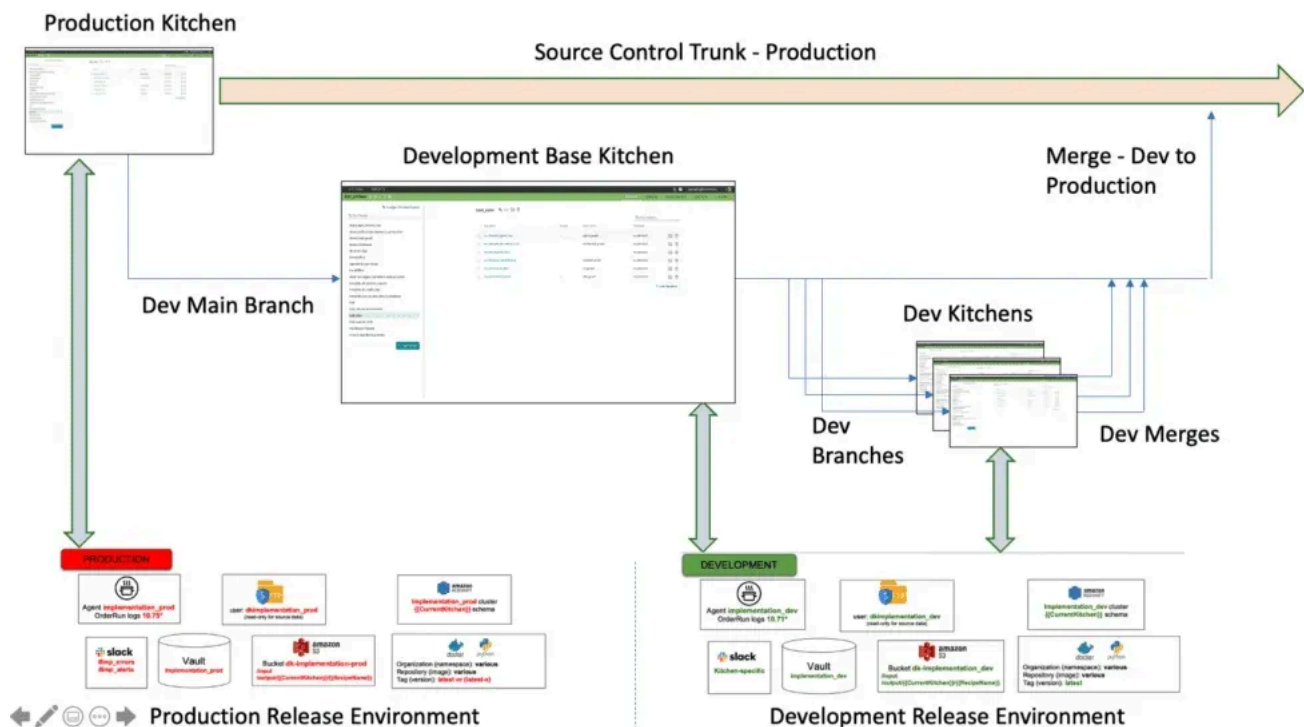
The same principle applies to AI agent environments. Each Claude session needs its own isolated view of the data and codebase, where it can read, write, experiment, and fail without any of that activity being visible to other sessions or to production. At DataKitchen, we call these isolated environments **Kitchens**. Each Kitchen bundles together three things Claude needs to work autonomously: a Git branch of the pipeline code, a pointer to the correct test data, and a separate compute environment. The test data itself is typically provisioned as a zero-copy clone of production, so Claude has a full representative dataset to run against without touching real data. Claude enters the Kitchen, writes or modifies a transform, runs it against the cloned dataset, evaluates whether the tests pass, and tries again if they do not. That loop repeats, unattended, until the work is done or Claude surfaces a genuine decision it cannot make alone. Production never sees any of it until the work is explicitly promoted.

Think of it as an ephemeral Kitchen for AI agents to work in. You can spin up many Kitchens for many agents simultaneously, each one cooking up a different approach to the same problem. Once their test cooking is complete, you evaluate the results, select the best dish, and promote it to production. The others get discarded. No mess, no risk, no cleanup.

Because Kitchens are fully isolated, there is nothing shared between them that could cause interference. A single engineer can run three or four Claude Code sessions in parallel, each on a different terminal, each working in a different folder, each mapped to a different Kitchen and Git branch, each taking a different approach to the same problem. They run unattended. Claude writes, tests, iterates, and either finishes or flags for human input. When you come back, you have multiple completed attempts to evaluate side by side rather than one sequential result. The orchestration is still yours: you set the brief for each Kitchen, they cook, you judge. But the execution is fully parallel and fully autonomous. The engineer's role shifts from writing pipeline code to directing a team of agents.

This isolation collapses the governance challenge down to a well-understood pattern: API-based access control. Instead of trying to audit every action an agent takes against a shared lakehouse, you control what the agent can do by controlling what the Kitchen API exposes. The agent can only see what it is given access to. It can only write to its own isolated context. The blast radius of any mistake is bounded by definition. **DataKitchen's DataOps Automation** platform provides this unified API, giving teams the infrastructure to run autonomous Claude sessions safely without rearchitecting their entire lakehouse from scratch. That way, Claude can try new SQL, a variation of Python Code, or a script tweak in isolation.

DataOps Automation gives you the process framework that sits on top of this. You branch your code and your environment simultaneously, so Claude works inside an isolated Kitchen with its own **Git branch**, test data pointer, and compute context while your main branch and production environment remain untouched. You treat data pipelines as processes rather than scripts, which makes it natural to hand off a bounded unit of work to an AI agent operating inside its own Kitchen. And you maintain full lineage of the action in an Order Run data structure, so every change Claude makes is traceable, reviewable, and revertable before anything touches production.



Governance of AI agents in data infrastructure is not mainly an AI issue. It is a concurrency issue. Address isolation first, and managing AI Agents becomes like overseeing multiple separate 'kitchens' of work. DataKitchen created its Kitchens so developers could work in isolated, aligned sandboxes and confidently combine their work. Fix the concurrency issue at the infrastructure level, and access control shifts from a complex organizational challenge to a simple API problem. This is crucial whether you have two developers or two hundred AI Agents.

Layer 2: FITT Architecture

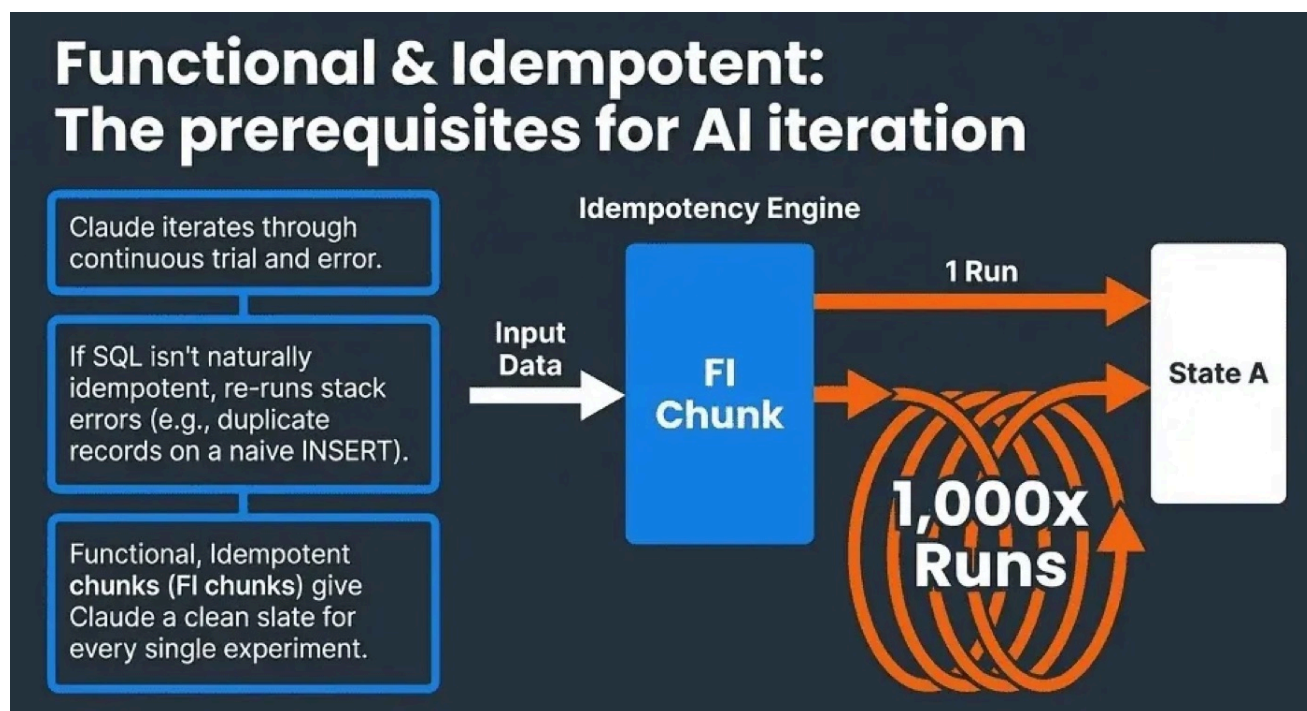
The second layer is architectural. We call it **FITT: Functional, Idempotent, Tested, Two-stage**. Most data architectures are designed around vendor capabilities or medallion-layer conventions. FITT is designed around data engineer productivity and, as it turns out, AI agent productivity too. The principles that make pipelines easier for humans to reason about and recover from are precisely the principles that make them safe and productive for an autonomous agent to work on.

Functional: same input, same output, every time

A functional transform has no mystery dependencies and no hidden state. For each code version, the transformation takes clearly defined inputs and produces predictable outputs. The same input produces the same output regardless of when or where it runs. Raw data is treated as immutable by default.

Functional Data Engineering eliminates an entire class of bugs that plague traditional data systems: no more wondering whether yesterday's transformation affected today's results, no more state-dependent edge cases that only surface in production.

For an AI agent, this property is transformative. Claude can reason about a functional transform completely from its definition. It does not need to understand the history of prior runs or the current state of upstream tables. It sees inputs, it sees the expected output shape, and it can write, test, and iterate on the logic in total isolation. The transform is a mathematical function. Claude can treat it like one.



Idempotent: run it a thousand times, get the same result

Idempotency is the property that changes how the whole team thinks about failure and iteration. When a pipeline is idempotent, recovery is trivial. Did something break due to a schema change? Fix the transform and re-run it. Getting unexpected results? Re-run with debugging enabled on yesterday's data

and compare. Want to test a change safely? Re-run it on a historical snapshot. This eliminates the paralyzing fear of corrupting production data that keeps junior engineers from experimenting and keeps senior engineers up at night.

The practical implementation challenge is that SQL is not naturally idempotent. A naive INSERT run twice creates duplicate records. A CREATE TABLE AS SELECT fails on a second run if the table already exists. Making SQL truly idempotent requires deliberate design: explicitly managing state, using parameterized time windows so transforms address a specific data snapshot rather than assuming current state, and structuring each transform as what the [FITT vs. Fragile SQL article](#) calls an 'FI chunk', a Functional and Idempotent chunk that becomes the fundamental building block of a FITT-compliant pipeline. Whether you are using dbt, Databricks, Snowflake, or BigQuery, the goal is the same: each chunk manages its own state, declares its dependencies explicitly, and can be safely re-executed at any time.

For Claude, idempotency is not a nice-to-have. It is the property that makes autonomous iteration possible at all. Claude's approach to any task is trial and error: write a solution, run it, evaluate the result, adjust, and try again. That loop only works if every run starts from a known state and produces a deterministic result. Without idempotency, Claude's re-runs stack errors on top of each other. With it, each iteration is a clean experiment.

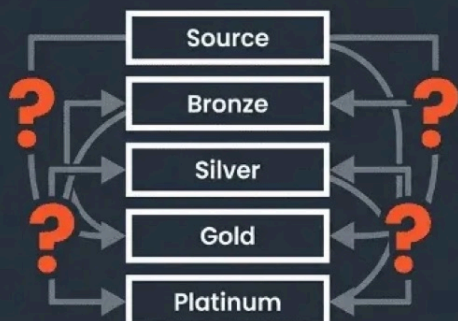
The FITT blog describes this as a 'build a little, test a little, learn a lot' development rhythm. That rhythm is exactly how Claude Code works. The architecture and the agent's working style are designed to work together.

Two-stage: raw to final, that is it.

Most teams inherit a multi-layered medallion architecture where bronze data does not match gold data, and debugging at 2 AM means tracing discrepancies across 5 intermediate layers. FITT replaces this with two stages: raw data that represents exactly what arrived from source systems, immutable and timestamped, and final data that is the complete analytical output rebuilt from that raw layer, no bronze, silver, gold, or platinum in between. The complete rebuild approach reconstructs the entire analytical dataset from immutable raw sources in a single orchestrated pass. If you need to verify last month's results or backfill three months of data after a bug fix, you adjust the time window parameters and re-run. The process is identical whether you are processing one day or one hundred.

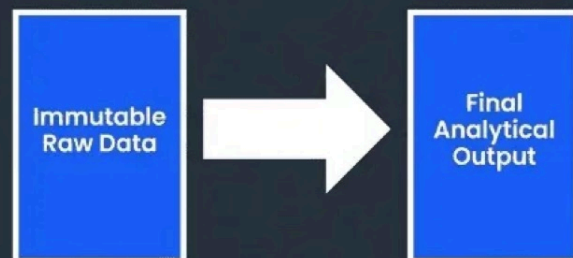
Two-Stage vs. Medallion: Bounding the problem for autonomous agents

The Medallion Maze



Multi-layered architectures force AI to navigate complex intermediate dependencies. Debugging requires tracing discrepancies across multiple fragile layers.

The FITT Contract



FITT enforces a complete-rebuild approach in a single orchestrated pass. Claude receives a clear contract: reconstruct final outputs directly from immutable raw sources.

This simplicity is a significant advantage for AI agents. A Claude session working inside a Kitchen does not need to understand which intermediate layer it is allowed to modify or which upstream tables are shared with other pipelines. It works with raw inputs and produces final outputs. The Two-stage contract is clear, bounded, and easy to test.

FITT is what makes Claude's problem-solving granular enough to succeed. An FI chunk has clear inputs, a clear expected output, and tests that define what 'done' means. Claude can pick one up, iterate on it in isolation without fear of breaking anything else, and know exactly when it has finished. That is the fundamental unit of autonomous AI data engineering.

The FITT architecture is similar to the [write-audit-publish pattern](#) that Netflix first talked about. Claude writes to its isolated Kitchen environment, audits the result against the test suite, and only promotes data forward when the checks pass. FITT provides autonomous quality control built into its architecture, not bolted on afterward.

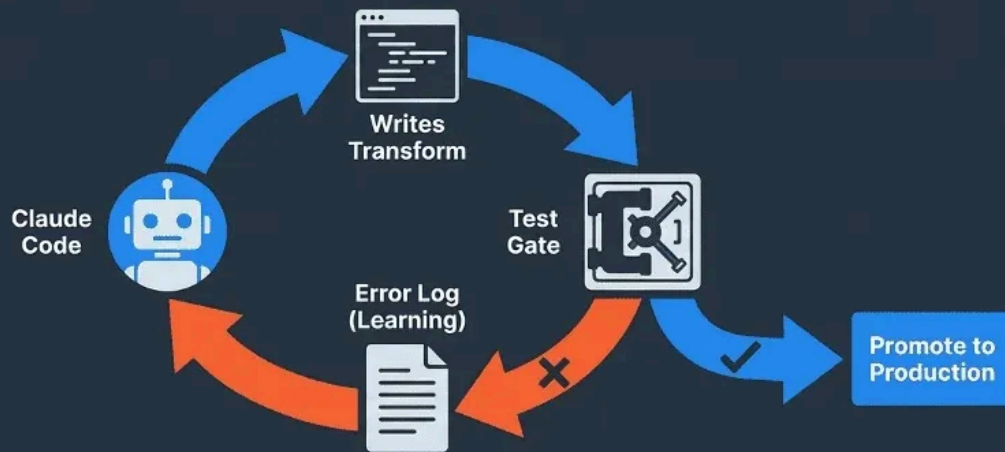
Layer 3: Data Testing

This is arguably the most important layer, and the most underinvested in most data teams.

Claude Code can generate code. It can run code. What it cannot do on its own is judge whether the output is correct. That judgment has to come from data tests. Without adequate test coverage, an AI agent has no feedback loop. It cannot learn from its mistakes, cannot verify its progress, and cannot know when it is done. Our [Definitive Guide to Test Coverage for Data Engineers](#) goes deep on what a real coverage strategy looks like across every layer of your pipeline.

Layer 3: Data Testing as the Autonomous Feedback Loop

AI can generate code, but it cannot judge correctness.



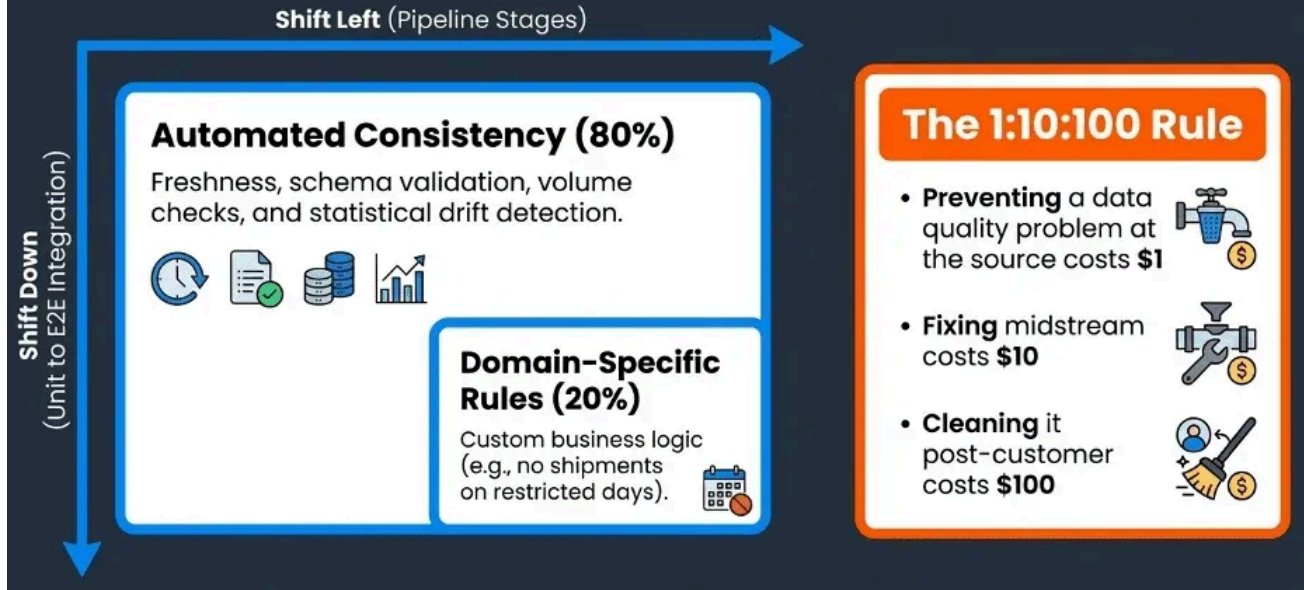
Claude writes a transform -> Runs the full test suite -> Reads the failure logs -> Adjusts approach -> Tries again.
Takeaway: Without data tests, a human must babysit every iteration. With tests, Claude self-directs for hours.

What adequate test coverage means in practice

The Definitive Guide frames test coverage around two complementary strategies: shift left and shift down. Shifting left means running data quality tests earlier in the pipeline lifecycle, catching problems before they propagate downstream and become exponentially more expensive to fix. The 1:10:100 rule captures the stakes: preventing a data quality problem at the source costs roughly \$1 per record; fixing it midstream costs \$10; cleaning it up after it has reached customers costs \$100. Shifting down means going beyond unit-level validation to integration and end-to-end testing that validates how components interact across the full pipeline, not just in isolation.

Comprehensive coverage operates across two distinct dimensions simultaneously. Production test coverage ensures that data quality remains high throughout live operations, catching freshness failures, volume anomalies, schema drift, and distribution changes as they happen. Development test coverage validates that code changes do not introduce regressions or break existing functionality when promoted from a Kitchen to production. Many tests serve double duty across both dimensions, functioning as both production monitors and development validators.

What adequate AI test coverage actually requires



In terms of what the tests themselves cover, automated consistency tests handle the bulk of the work, roughly 80% of what a comprehensive suite needs. These include freshness and volume checks, schema validation, referential integrity, and data drift detection using statistical methods to catch gradual shifts in data characteristics that simpler threshold-based alerts miss. The remaining 20% is custom domain-specific tests that encode business rules no automated system can infer from data patterns alone: verifying that a medical practice has no more appointments than it has doctors, ensuring no shipments occur on restricted days, and confirming that revenue figures fall within historically expected ranges. Those business rule tests are where your domain knowledge becomes part of the quality gate that Claude must pass.

Achieving Test Coverage: 7.2 Months of Engineering vs. 2 Steps of Automation

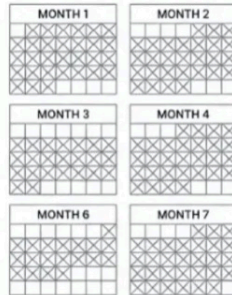


THE MANUAL GRIND

30 Minutes per test
1,250 Total Hours coding
156 Days (8 hrs/day)
31 Weeks (5 days/week)

7.2 MONTHS

With no time for meetings, breaks, or vacations.

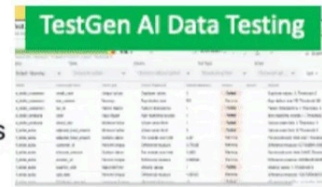


Number of Tests	2,500		
Number of Tables	20	30	
Number of Columns	1,000	1,250	
Minutes per Test		156	8 hours/day
Days		31	5 days/week
Weeks		31	4.35 weeks/month
Months		7.2	



THE DATAOPS ADVANTAGE

Profile Data
Generate Tests



2 STEPS

2,500 tests generated immediately.

TestGen compresses **156 days** of senior engineering work into **minutes**.

Representative test data is equally important. Claude needs to run against data that actually exercises the edge cases and failure modes of your pipeline. Minimal or synthetic test data can lead to false confidence. The zero-copy clone in the Kitchen exists precisely to give Claude a full, realistic dataset to work against without risking production. Development like this: yesterday's data run against today's code.

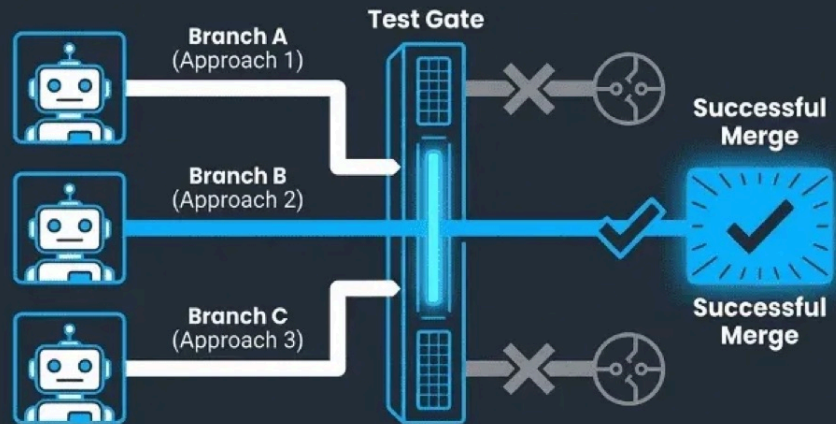
Tests as the AI feedback loop

Here is the workflow that makes long autonomous Claude sessions possible: Claude writes or modifies a transform, runs the full test suite against the branch, reads the failure output, adjusts its approach, and tries again.

Prompting Discipline and Parallel Agent Workflows

Prompting Discipline

- Ask for a plan before coding.
- Demand minimal changes.
- Force Claude to write a test for every single modification.



Treat Claude sessions like concurrent developers. Spin up multiple branches to test different technical approaches simultaneously. Merge the one that passes FITT tests first. Discard the rest.

This loop can run for 20, 30 minutes, or several hours when the problem is complex. You are not babysitting it. You check back in, and either the tests are green, or Claude has surfaced a real issue that needs guidance on.

High coverage data tests are the mechanism that turns Claude Code from a code generator into an autonomous engineering agent. Without them, you need a human in the loop for every iteration. With them, Claude can self-direct.

Data testing is how you manage AI risk. If Claude goes down the wrong path for 30 minutes, the tests catch it. Nothing ships until the tests pass. Your test suite is your safety net and your teacher simultaneously. DataKitchen's open-source [TestGen](#) is built exactly for this: it generates thousands of data quality tests automatically from your schema and data profile, giving Claude the feedback loop it needs to work autonomously without requiring your team to hand-write every test case.

For a comprehensive treatment of how data testing and observability fit together in a modern data engineering practice, see our new book [The DataOps Way to Data Quality and Data Observability](#).

Tips: How Data Engineers Work with Claude Code

The first thing to understand about Claude Code is that it is a CLI tool, not an IDE plugin, and that distinction matters. Claude Code manages its own persistent memory through `CLAUDE.MD` files that are stored in your project folders. These files are loaded automatically at the start of every session, so Claude picks up exactly where you left off without you having to re-explain your schema, conventions, or the business rules you spent 20 minutes describing last Tuesday. Every AI chat session starts with a ticking clock as context builds and earlier information is pushed out. `CLAUDE.md` files solve this. Use them

aggressively. Ask Claude to document what it has built and what assumptions it is making so that knowledge survives session restarts, machine changes, and team handoffs. Version control those files in Git alongside your pipeline code.

Operating the Engine: Mastering the Claude Code CLI

CONTEXT PERSISTENCE

Claude Code is a CLI tool, not an IDE plugin. Aggressively use `[CLAUDE.md]` to persist memory, schemas, and business rules across sessions. Version control these files.

CONTEXT FLUSHING

Watch the context usage bar. Execute `[/clear]` at 75% to prevent degradation.

SESSION MANAGEMENT

Use `[--continue]` or `[--resume]` to effortlessly manage scoped pipeline folders. Keep PRD documents updated dynamically.

Context management is a skill you will develop quickly. Watch the context usage bar in the CLI, and use `/clear` when it reaches 75%. Organize your work into one session per folder, grouped by topic or pipeline stage. When you need to resume, the fastest path is `claude , continue` to pick up the last session instantly, or `claude , resume` to browse and select from named sessions. For longer-running agent workflows, create a PRD (planning document) in the folder; ask Claude to update it as work progresses. A well-maintained PRD is structured knowledge, not just chat history, and it is what allows a teammate or a fresh Claude session to step in without losing ground.

Prompting discipline makes a measurable difference. Before Claude starts any task, ask it to make a plan and review it with you before touching anything. Ask what assumptions it is making. This two-minute check prevents Claude from spending 40 minutes on an approach that misses a constraint you forgot to mention. During work, push Claude toward minimal changes: the prompt that consistently produces tighter, more reviewable diffs is simply asking for the minimum change needed to achieve the goal. After Claude makes changes, ask it to write a test that proves the changes are correct. This directly feeds the test coverage that makes autonomous iteration possible and keeps the feedback loop tight. Explicitly save work to disk before clearing context.

The highest-leverage pattern in the slides is parallel development. Think of each Claude session as a separate developer. Different folders mean different branches, and you can run multiple terminals simultaneously, each with Claude grinding on a different pipeline unit in isolation. You do not need to make perfect upfront decisions about which approach is right. Spin up two branches with two different

approaches, let Claude iterate on both, and merge whichever one passes the tests first. The mental model shift is significant: you move from writing pipeline code to directing a team of autonomous agents, each working in its own safe sandbox, with data tests as the shared definition of done. Four-hour tasks become 15-minute tasks. The focus shifts entirely to what needs to be built, not how to build it.

A Walkthrough Example

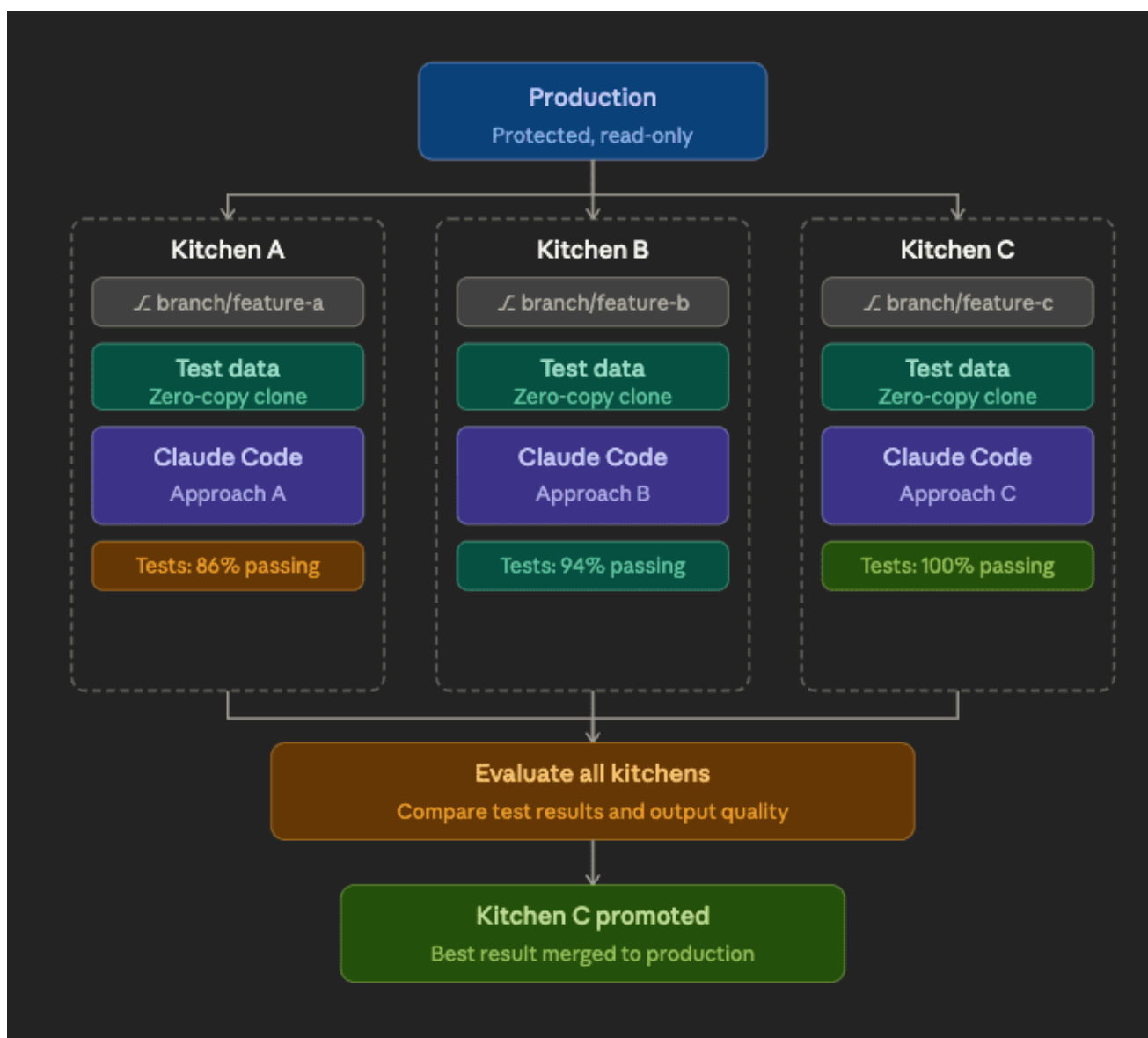
A data engineer has a task: rewrite a slow, brittle SQL transform that ingests raw sales data into a final reporting table. The existing version breaks on reruns, has hidden state dependencies, and takes 40 minutes to run on a bad day. Rather than working through one approach sequentially, the engineer spins up three approaches and lets Claude explore three strategies in parallel: parameterized time windows, explicit staging-table logic, and a full-step rebuild from raw.

Provisioning the Kitchens

Each Kitchen is a bundle of three things: a Git branch forked from main, a zero-copy clone of the production dataset, and a separate compute context.

The zero-copy clone provides each Claude session with a complete, realistic, production-scale dataset to work with without duplicating storage. Each Kitchen references the same underlying production data blocks, with a copy-on-write layer on top. When Claude generates output, those changes are made to the Kitchen's isolated layer. Production data blocks remain unchanged. The cost is nearly zero. The isolation is thorough.

The engineer opens three terminal sessions, points each at a different folder mapped to a different Kitchen, gives each a brief, and walks away.



Claude Iterates in Each Kitchen

Inside each Kitchen, Claude reads the CLAUDE.md context file, inspects the schema, and starts writing. It runs the transform against the cloned dataset, evaluates the output against the test suite, reviews any failures, revises, and reruns. It does not wait for input. It repeats this process until the tests pass or it encounters a decision it cannot make on its own. Kitchen A, using the parameterized time-window approach, encounters problems with late-arriving records. After four iterations, it reaches 86%, but notes that one cross-period consistency check is still failing and may require human judgment on an acceptable tolerance. Kitchen B, employing the staging table approach, proceeds smoothly through the standard automated tests and completes at 94%. It records that the remaining failures are mainly in a single edge case involving partially fulfilled cancelled orders. Kitchen C, running the full step rebuild, takes the most aggressive route. All tests pass: 100%. It logs an 8-minute rebuild time on the cloned dataset.

Evaluation and Promotion

With test results in hand across all three Kitchens, Claude compares the outcomes. Kitchen A is eliminated: below threshold and flagged for a structural issue that would require rearchitecting the windowing logic. Kitchen B passes at 94%, but it carries a known edge-case failure that would require additional work to resolve. Kitchen C wins: 100% test pass rate, clean output, and an 8-minute rebuild cycle, well within the production schedule for a transform running every 15 minutes.

Kitchen C's branch is promoted. The engineer reviews the diff, a clean, readable transformation built on a full rebuild from immutable raw sources following FITT conventions, and merges it into main in ten minutes. Kitchen A and Kitchen B are discarded. Their branches are deleted, their compute contexts shut down, and their zero-copy clones released. Nothing from those sessions ever made it into production.

What Just Happened

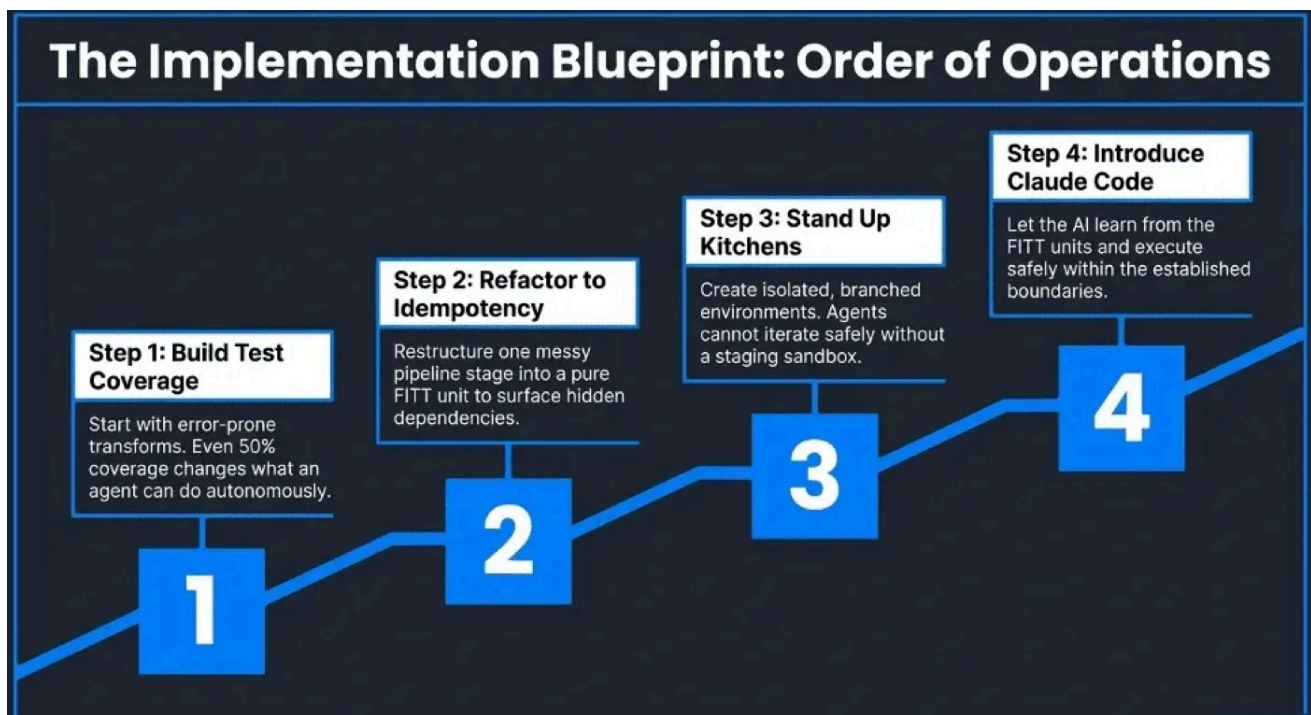
A task that would have taken a full day of sequential engineering was completed in under two hours, with three approaches explored simultaneously and the engineer's active involvement limited to writing the briefs and reviewing the winning diff. The test suite did the continuous validation. Claude did the iteration. The engineer made the final call. That division of labor is where the 10x comes from.

The 10x Is Real, But It Is Not Magic

We want to be specific about what 10x means here. It does not mean that Claude Code writes 10x as many lines. Tasks that used to take a day take two hours. A single engineer can run multiple parallel workstreams across isolated Kitchens simultaneously. Debugging time drops sharply because tests catch errors before they propagate. Onboarding new pipeline logic is accelerated because data tests are automatically generated and document the expected behavior. And code review is faster because every diff is scoped to a FITT unit with a clear, bounded surface area.

None of this happens without the foundation. DataOps provides the branching and process discipline. FITT provides the granularity and idempotency that make autonomous iteration safe. Data testing provides the feedback loop that enables autonomous iteration.

Remove any one of the three, and the productivity gains collapse. Together, they multiply each other.



How To Get Started Today

How do I start moving toward this model that enables Claude Code's success? Build your data test coverage first. Start with the transforms that are most error-prone or most business-critical. Even 50% coverage on key transforms changes what Claude can do autonomously. Then refactor toward idempotency. Pick one messy pipeline stage and restructure it as a FITT unit. The refactor itself will surface hidden dependencies and assumptions that have been buried in the code. Next, stand up isolated branched environments. If your pipeline can only run in one place, you cannot safely let Claude iterate. Then, and only then, introduce Claude Code. With the foundation in place, the onboarding is fast. Claude learns the patterns from your existing FITT units and tests, and productive output starts immediately.

Conclusion

AI coding tools are going to change data engineering. That much is certain. The question is whether the change for your team is a genuine productivity multiplier or just a faster way to generate untested code that breaks in production.

The Reality of 10x Data Engineering



- 10x does not mean AI writes 10x more lines of code. It means a 4-hour task collapses to 15 minutes.
- You shift entirely from writing pipeline code to directing a team of autonomous agents, using data tests as your shared definition of done.

The engineers getting 10x results are not the ones with the best prompts. They are the ones who invested in DataOps discipline, FITT architecture, and data test coverage before they picked up an AI tool. Claude Code is the accelerant. The foundation is what makes the acceleration safe.

The 10x opportunity from LLMs is not limited to data engineers. Analysts working directly with AI to answer business questions face the same foundational challenge: AI is an amplifier, and if what sits beneath it is untrustworthy, poorly documented, or missing business context, it can amplify bad answers at speed. We have a companion framework for that challenge in [The Equation for AI Success: DT + DX + CTX = 10x](#). Where this post addresses how data engineers build the pipelines that make AI safe and productive, that one addresses how analysts make their data legible, trusted, and context-rich enough for AI to reason over it reliably. The two equations are meant to work together.

Opportunity to use AI to 10x the productivity of:

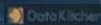
The Integration Ecosystem
Data Engineers who are integrating the data

DataOps + FITT + Data Testing = 10x

The Insight Ecosystem
Data Users directly using the data to give business insight

DT + DX + CTX = 10x

The architectural foundation is what makes the acceleration safe across the entire enterprise.



Build a Data Mesh Factory with DataOps

Read the original article at: <https://datakitchen.io/white-papers/build-a-data-mesh-factory-with-dataops/>



Introduction to Data Mesh

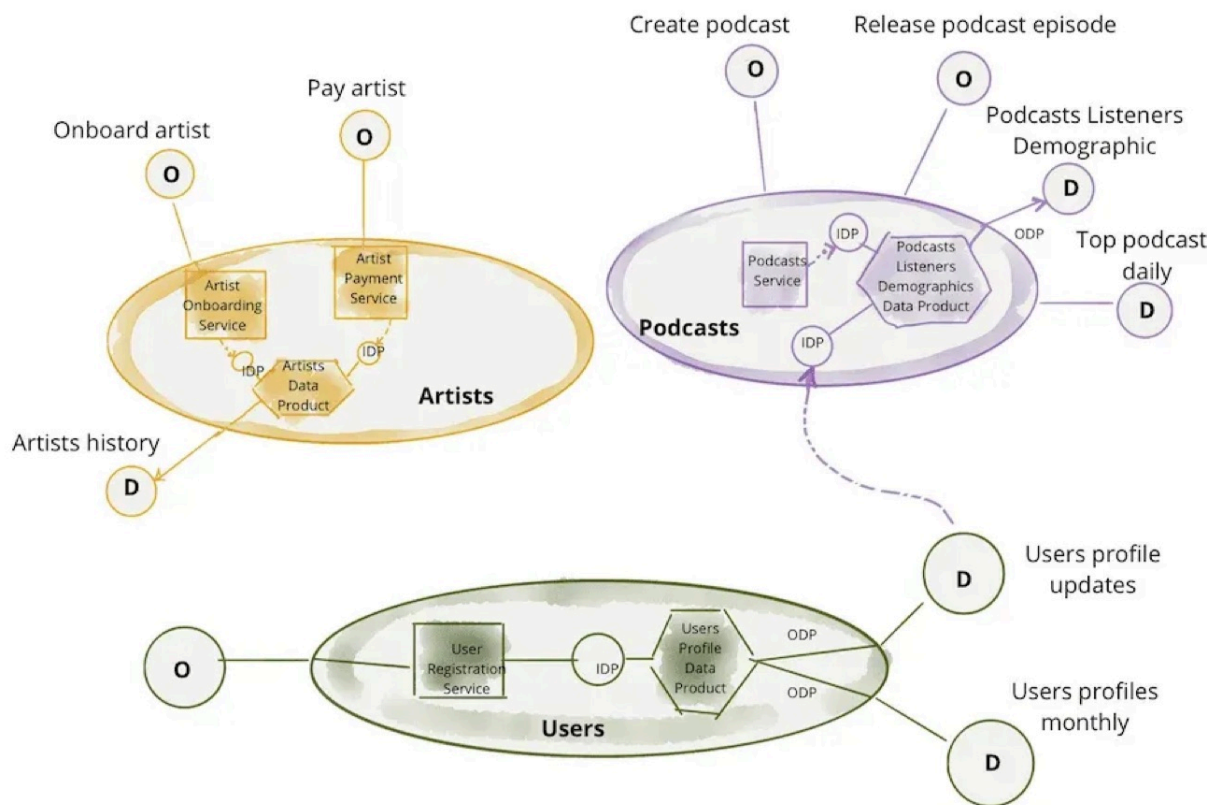
The data mesh addresses the problems characteristic of large, complex, monolithic data architectures by dividing the system into discrete domains that are managed by smaller, cross functional teams. Data mesh proponents borrow the term “**domain**” from the software engineering concept of “**domain-driven design (DDD)**,” a term coined by Eric Evans. DDD divides a system or model into smaller subsystems called domains. Each domain is an independently deployable cluster of related microservices which communicate with users or other domains through modular interfaces. Each domain has an important job to do and a dedicated team - five to nine members - who develop an intimate knowledge of data sources, data consumers and functional nuances. The domain includes data, code, workflows, a team, and a technical environment. Data mesh applies DDD principles, proven in software development, to data analytics. If you’ve been following DataKitchen at all, you know we are all about transferring software development methods to data analytics. **The concept of data mesh**, proposed by **Zhamak Dehghani**, has taken the industry by storm. We’d like to add a discussion of how data mesh lifecycle management requires DataOps.

Below we’ll look at an example of a data architecture partitioned into domains to illustrate data mesh.

You may have used a media streaming application such as Spotify or SoundCloud. A streaming service has to handle a variety of activities that logically partition into different areas:

- Artists - onboard, pay, manage, ...
- Podcasts - create, release, play, ...
- Users - register, manage profiles, manage access, ...

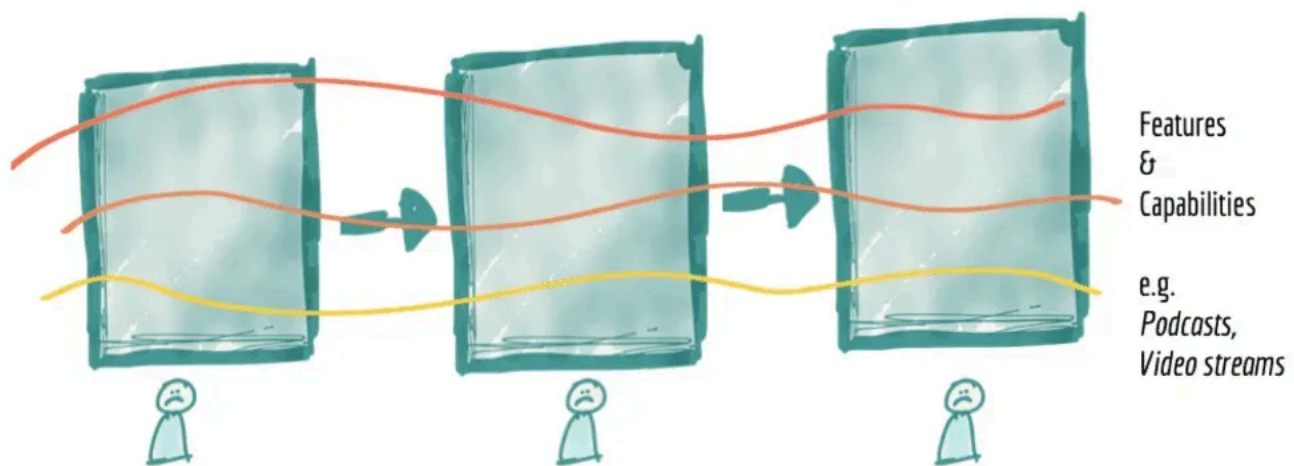
A centralized enterprise application would combine all of these functions and services into one monolithic architecture. A data mesh tackles each functional domain as a separate set of services. In figure 2 below, three domains are shown. A dedicated team handles artists, podcasts and users respectively, but each domain depends on data and services from other domains and operational systems. The domains in the figure are shown receiving data from input data ports (IDP) and transmitting data to output data ports (ODP). The act of onboarding a new artist activates an artist onboarding service that updates the artists domain. Perhaps when a user plays a podcast (within the podcast domain) it triggers an input into the artist's domain, which activates the artist payment service.



Application scalability improves as each group independently manages its domain. For example, the users domain team could add a new data set or new microservices related to users data with complete autonomy.

Improve Agility with a Data Mesh

One difficulty of large, centralized enterprise architectures is that they don't lend themselves to using Agile methods. The centralized architecture consists of myriad pipelines that ingest, process and serve data. Everyone on the data team works on their specialized part of the larger system, and they rely upon emails, documents and meetings to stay organized. Figure 3 shows the strained lines of communications that cross functional boundaries. Roles tend to be more specialized on a big team, so there are bottlenecks. With heavyweight processes, it becomes much more bureaucratic to make a change to any part of the architecture. With a steady stream of requests for new data sources and new analytics, the centralized team managing the platform can quickly exceed their capacity to keep up. Customers are on a journey to get insight, and they may not know exactly what they want until they see it. With large systems, it's much harder to iterate toward a solution that addresses a user's latent requirements.



Data mesh decouples the domain teams from each other. By partitioning the system into pieces, each domain team is able to work uninterrupted and at their natural iteration cadence. Over time, domain teams attain a much more intimate understanding of their data sources and uses, leading to more effective technical solutions. The data team addresses the backlog of tasks more quickly because task coordination is simpler with a smaller team. The data mesh team assumes total lifecycle ownership of the domain.

The domain is their internal “product” and the domain product manager is their empowered mini-CEO. You may have heard the term “product thinking” with respect to a data mesh. Success means excellently serving the needs of internal customers. Team members cover all the roles, so there are more hybrid players enabling more flexibility in responding to surges in demand for a particular skill set. When something goes wrong, the domain team has all of the right incentives to iterate on a solution.

The organizational concepts behind data mesh that we have covered are summarized as follows.

- Five to nine-person team owns the dev, test, deployment, monitoring and maintenance of a domain.
- The team organizes around the domain, not the underlying toolchain or horizontal data pipelines.

- Domain data assets, artifacts and related services are viewed as the team's product. The product includes data and operations. Data consumers are the domain team's customers.
- Data Engineers must develop an intimate understanding of data sets to really add value.

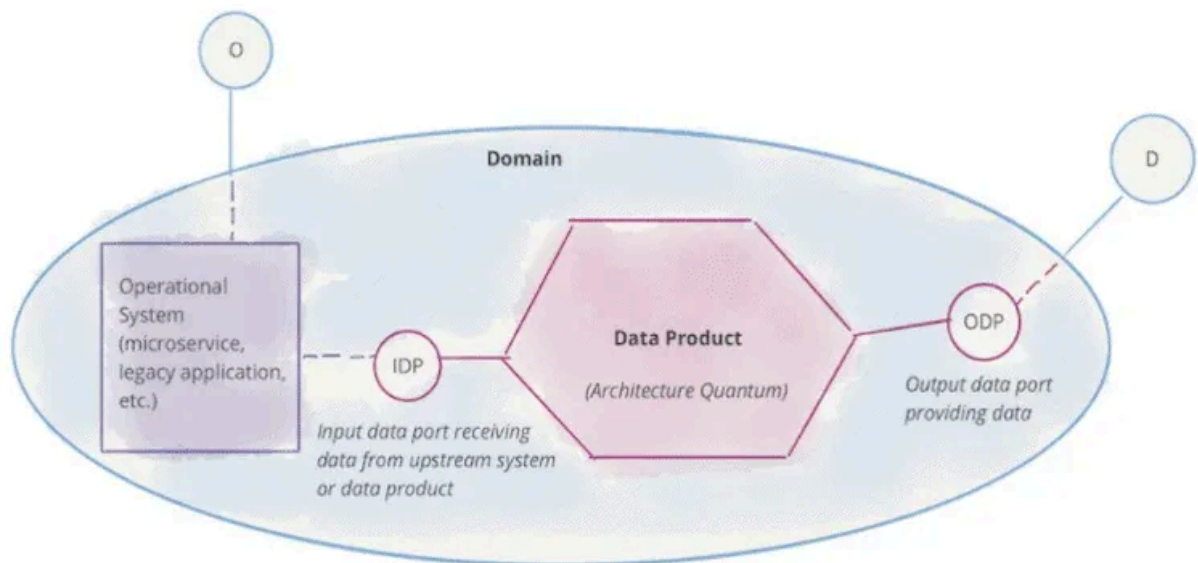
Technical Side of Data Mesh

We've talked about data mesh in organizational terms and how team structure supports agility. Let's take a deeper look at [the technical side of data mesh](#) before we provide an application example.

The data industry has a wide variety of approaches and philosophies for managing data: Inman data factory, Kimball methodology, star schema, or the data vault pattern, which can be a great way to store and organize raw data, and more. Data mesh does not replace or require any of these.

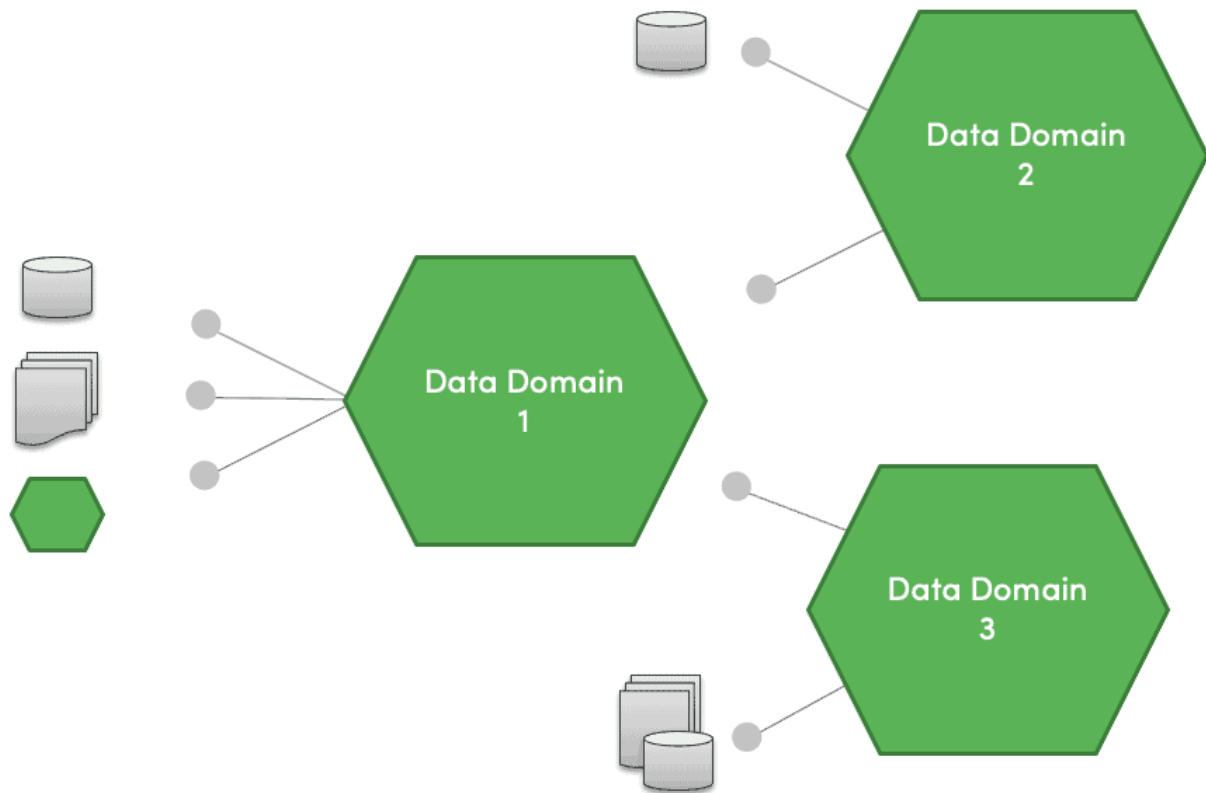
The data mesh is focused on building trust in data and promoting the use of data by business users who can benefit from it. In essence, a domain is an integrated data set and a set of views, reports, dashboards, and artifacts created from the data. The domain also includes code that acts upon the data including tools, pipelines, and other artifacts that drive analytics execution. The domain requires a team that creates/updates/runs the domain, and we can't forget metadata: catalogs, lineage, test results, processing history, etc, ...

Instead of having a giant, unwieldy data lake, the data mesh breaks up the data and workflow assets into controllable and composable domains with inherent interdependencies. Some domains are built from raw data. Other domains are built from raw data and the output of other domains. Figure 4 below shows a simplified diagram of a domain receiving input data from an upstream source like an operational system (O) and supplying data (D) to a customer or consumer.

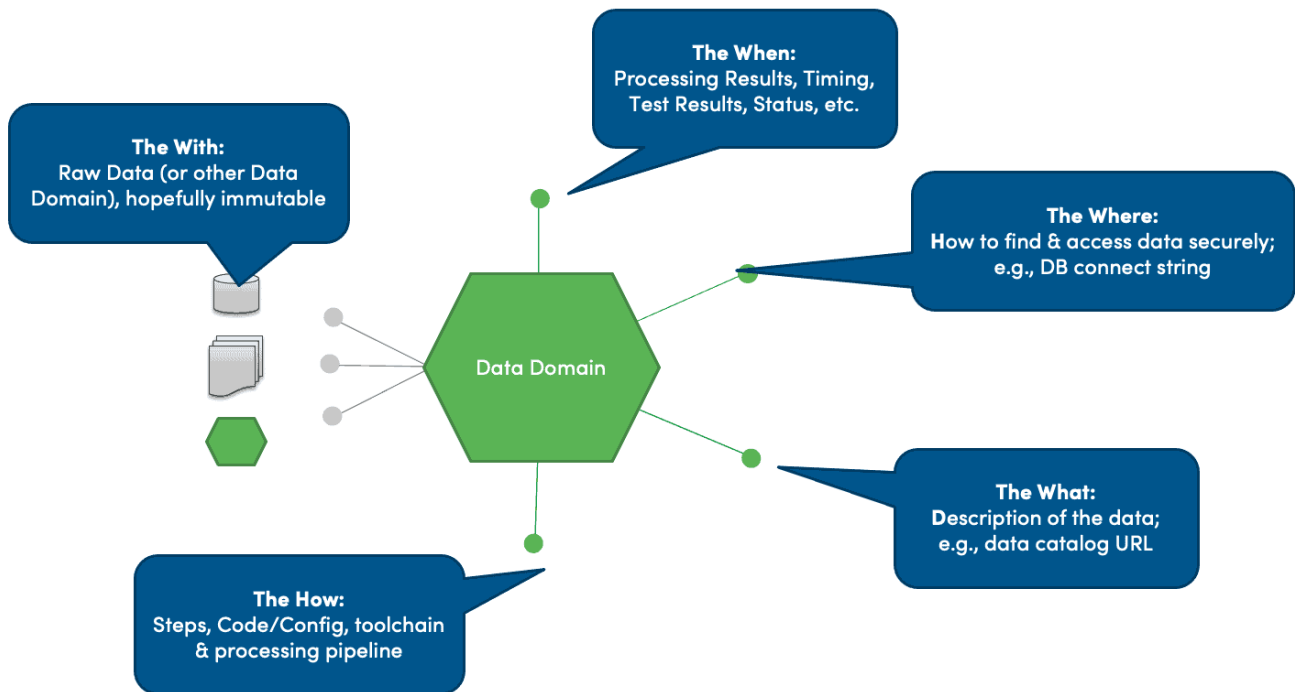


There's a clear order-of-operations challenge in building systems based on interdependent domains. For example, in the domain diagram in Figure 5 below, imagine that domain 1 has a list of mastered customers, which is utilized by domains 2 and 3. There's an implied producer-consumer, order of

operations relationship between these two. It's also very important to make domains composable and controllable.



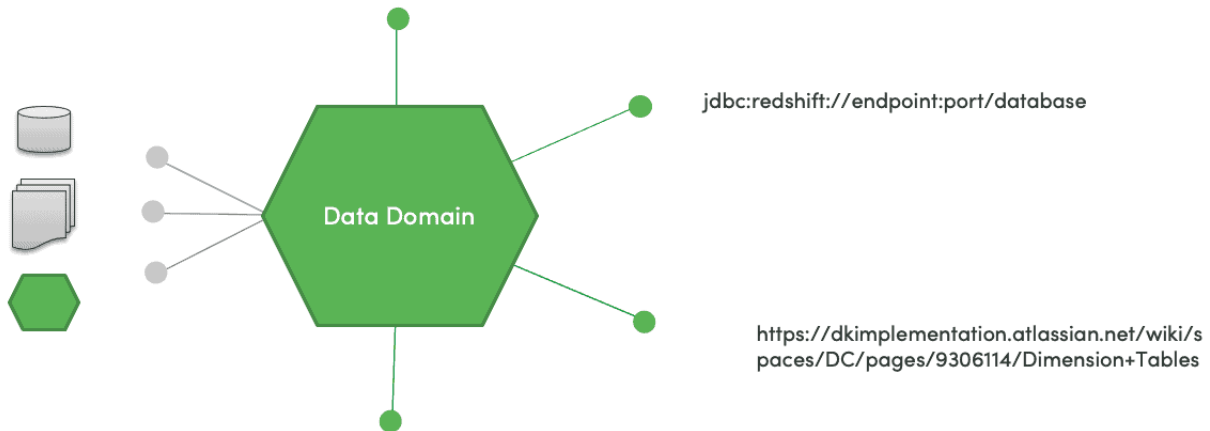
The consumers and customers of a domain can look at it as a black box. They don't have to think about the domain's internal complexity. The domain is accessed through its external interfaces which can be easily remembered as the four "W's" and an "H:"



- **What** is in the domain - description of the data, e.g., data catalog URL
- **Where** is the data - how to find and access the data securely, e.g., DB connect string
- **When** was the domain created/tested - data process lineage including artifacts like processing results, timing, test results, status, logs, etc., ...
- **With** what components was the domain created, set of raw data (or other data domain) used, hopefully immutable
- **How** was the domain created - steps, code/configuration, toolchain, and processing pipeline used to build the data

It's convenient to publish a set of URLs that provide access to domain-related data and services. How does one get access to a domain? How does one edit the catalog? How does a customer get status? Where is the source code that was used? A published set of URLs that access domains allow your team to better collaborate and coordinate. Figure 7 shows a domain with associated URLs that access key tools and orchestrations.

<https://cloud.datakitchen.io/#/orders/dc/Production/runs/60e82aa8-2518-11eb-8653-c2e92ba8ebec>



<https://cloud.datakitchen.io/#/recipes/dc/Production/agile-analytic-ops/variations/prod-env-DevSprint-build-now>

Benefits of a Domain

A well-implemented domain has certain attributes that offer benefits to the customers or domain users:

- **Trusted** - Users wish to be confident that the data and artifacts are correct. Trust must be earned, which is why it is so important for a domain to have interfaces that enable introspection and access. Users should be able to ask the domain questions and get answers.
- **Usable by the teams' customers** - Customer experience improves by virtue of having a dedicated team that attains intimate knowledge of the data and its use cases. Also, the domain must support the attributes that are part of every modern data architecture. **Discoverable** - users have access to a catalog or metadata management tool which renders the domain discoverable and accessible. **Understandable and well-described** - terms are defined in a dictionary and the domain has clean, well-designed interfaces. **Secure and permissioned** - data is protected from unauthorized users. **Governed** - designed with **data quality** and management workflows that empower data usage. **URL/API Driven** - can easily interoperate with other domains through a hyperlink (URL) or application programming interface (API).
- **Clear accountability** - users interact with a responsive, dedicated team that is accountable to them. Easy to report problems and receive updates on fixes. Users may request new insights/improvements and get them into production quickly.
- **Discoverable** - users have access to a catalog or metadata management tool which renders the domain discoverable and accessible.
- **Understandable and well-described** - terms are defined in a dictionary and the domain has clean, well-designed interfaces.
- **Secure and permissioned** - data is protected from unauthorized users.
- **Governed** - designed with **data quality** and management workflows that empower data usage.

- **URL/API Driven** - can easily interoperate with other domains through a hyperlink (URL) or application programming interface (API).
- Easy to report problems and receive updates on fixes.
- Users may request new insights/improvements and get them into production quickly.

Organizing a data architecture into domains is a first-order decision that drives organizational structure, incentives and workflows that influence how data consumers use data. Domains change the focus from the data itself to the use cases for the data. The customer use cases in turn drive the domain team to focus on services, service level agreements (SLA) and APIs. Domains promote data decentralization. Instead of relying on a centralized team, where the fungibility of human resources is an endless challenge, the domain team is dedicated and focused on a specific set of problems and data sets. Decentralization promotes creativity and empowerment. In the software industry, there's an adage, "you build it, you run it." Clear ownership and accountability keep the data teams focused on making their customers successful. As an organizing principle, domains favor a decentralized ecosystem of interdependent data products versus a centralized data lake or data warehouse with all its inherent bottlenecks.

Data Mesh in a Real-World Pharmaceutical Application

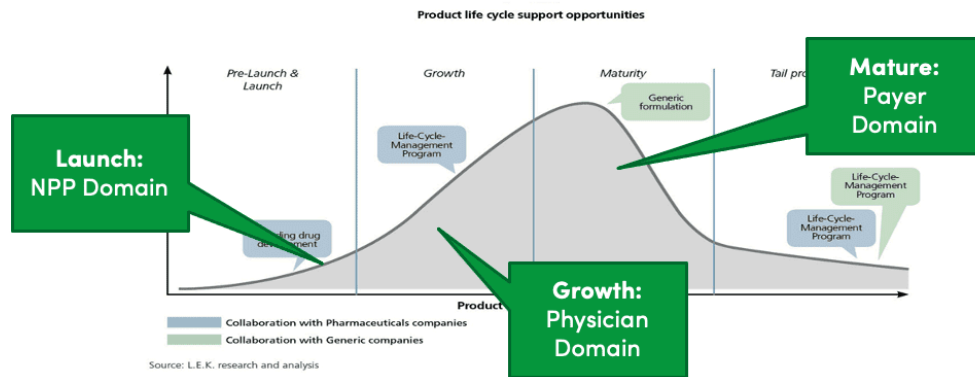
For those embarking on the data mesh journey, it may be helpful for us to discuss a real-world example and the lessons learned from an [actual data mesh implementation](#). Some of our senior staff have extensive experience using the data mesh design pattern with pharmaceutical company data. In the United States, private manufacturers of pharmaceuticals receive a patent for a limited period of time - approximately 20 years. In figure 8 below, we see that the data requirements are quite different for each of three critical phases of a drug's lifecycle:

Lifecycle Phase	Marketing Target	Data Focus
Launch	Patients	Non-Personal Promotion: emails, website visits, even radio ads
Growth	Physicians	Sales, claims data, anonymized patient data
Mature	Payer / Insurance Co	Rebates, formulary

Table 1: Lifecycle phases of pharmaceutical product launch

Each distinct phase of the drug lifecycle requires a unique focus for analytics. During the launch phase, the focus is on marketing to patients through consumer channels. A successful launch lays the groundwork for the growth phase where physicians prescribe the drug (the manufacturer hopes). As generic alternatives become available, the market passes through the maturity phase where cost efficiency and margins become paramount. There are different teams within the pharmaceutical company that focus on the respective target markets.

Commercial Pharma Analytics



US Commercial Pharma Domains

- **NPP** (Non-Personal Promotion): emails, web site visits, even radio ads
- **Physician**: doctor (& other outlets) sales, claims data, anonymized patient data
- **Payer**: Payer/Plan, rebates, formulary

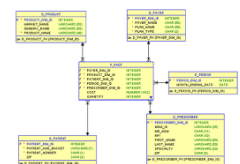
Drug companies maintain internal data sets, but also rely heavily upon third-party data (a multi-billion dollar industry). Some data sets are used by multiple teams, but that introduces complexity. Figure 9 shows the wide array of data sets used by pharmaceutical product launch organizations.

What About the Data?

Sub-national Weekly data
 Sub-national Payer Data
 Sub-national Institutional (DDD) Data
 National Prescription Audit Data
 Sales Force Alignment Data
 Longitudinal Patient Data
 Sub-national Profit and Loss Data
 Sub-national Claims and Co-pay Data
 Payer and Plan Formulary Data
 Census Data
 Stocking Data
 Source of Business
 AMA Data
 Retail OTC Data
 Buy and Bill Data
 Field Calls and Promotional Activity Data
 Rep Expenses and Vacancy Data
 Hotline Verification Data
 Contract and Payer Rebates Data
 Veeva CRM Data
 ERP Data
 NPP Data
 Forecast Data
 Primary Research Data

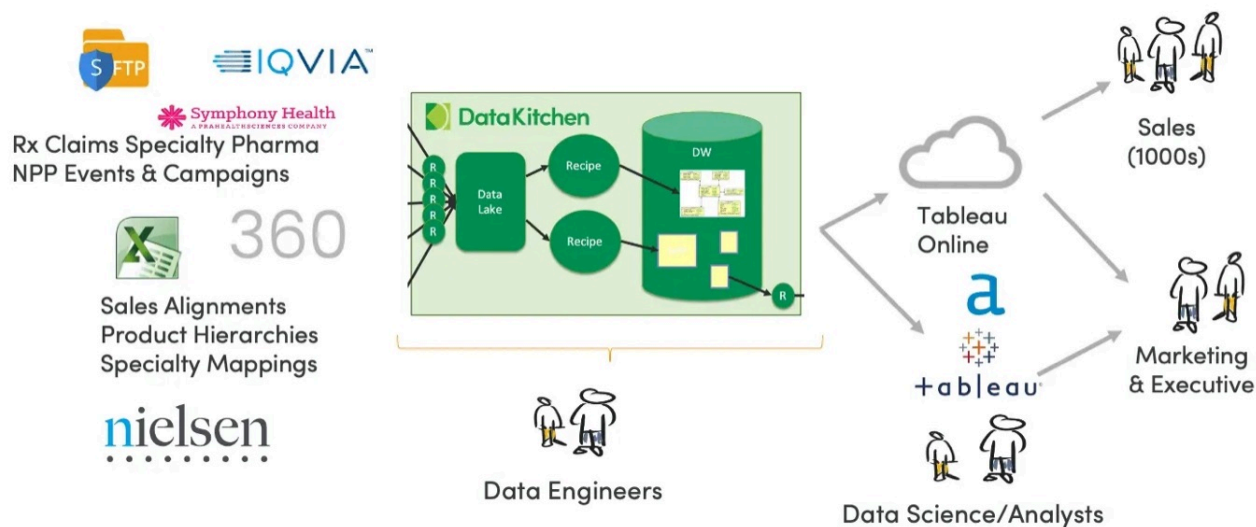
What about the data in each domain?

- Each domain has separate data sources
- Overlapping entities (e.g., physicians) exist in each domain
- Each domain has different cycle times of product (i.e., daily, weekly, hourly, etc.)
- Each data domain has its unique characteristics.
- For instance, subnational physician data from IQVIA – purchased by pharma companies – may not 1:1 match claims data, which may not match payer data. This is due to data supplier issues & timing projection algorithms.

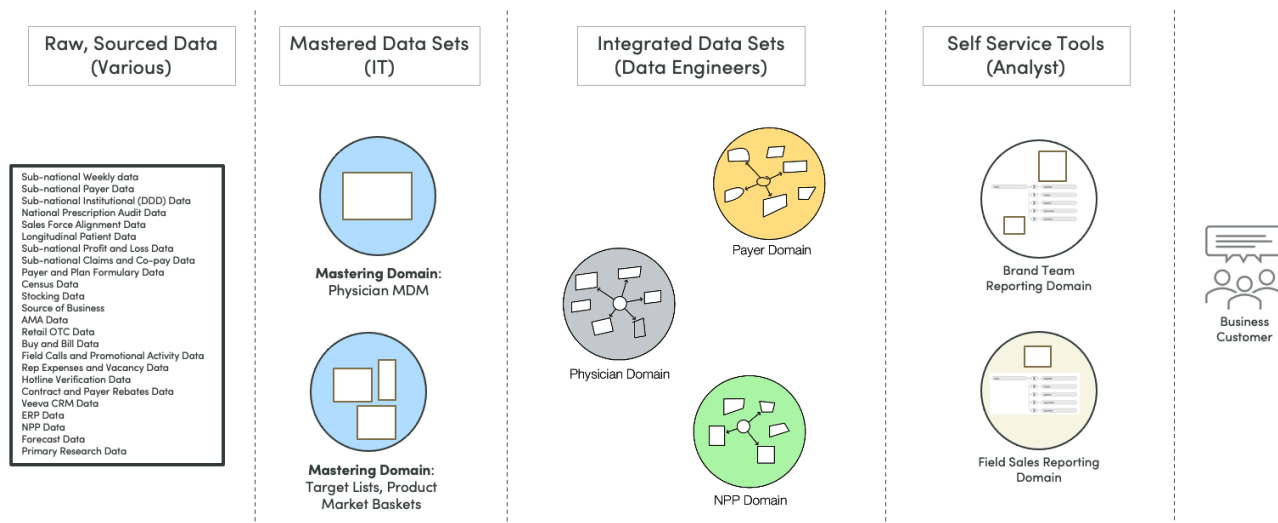


Some data sets are a mix of actual and projected data, complicating their use with other data sets that purport to be the same thing, but use a different algorithm to fill in gaps or projections. Two data sets of physicians may not match. They each tell a different story about the data.

Figure 10 shows an example processing architecture with data flowing in from internal and external sources. Each data source is updated on its own schedule, for example daily, weekly or monthly. DataKitchen DataOps Automation ingests data into a data lake and runs recipes to create a data warehouse that is leveraged by users and self-service data analysts. These users are very important to the company because their success greatly influences the strength of the revenue ramp in the drug's growth and maturity phases.



Let's consider how to break up our architecture into data mesh domains. In figure 11 we see our raw data shown on the left. First, the data is mastered, usually by a centralized data engineering team or IT. Each of the mastered data sets could be a domain. For example, there may be one million physicians in the US, but for a given drug, perhaps only 40,000 are important. Getting this standardized is important because it affects sales compensation.

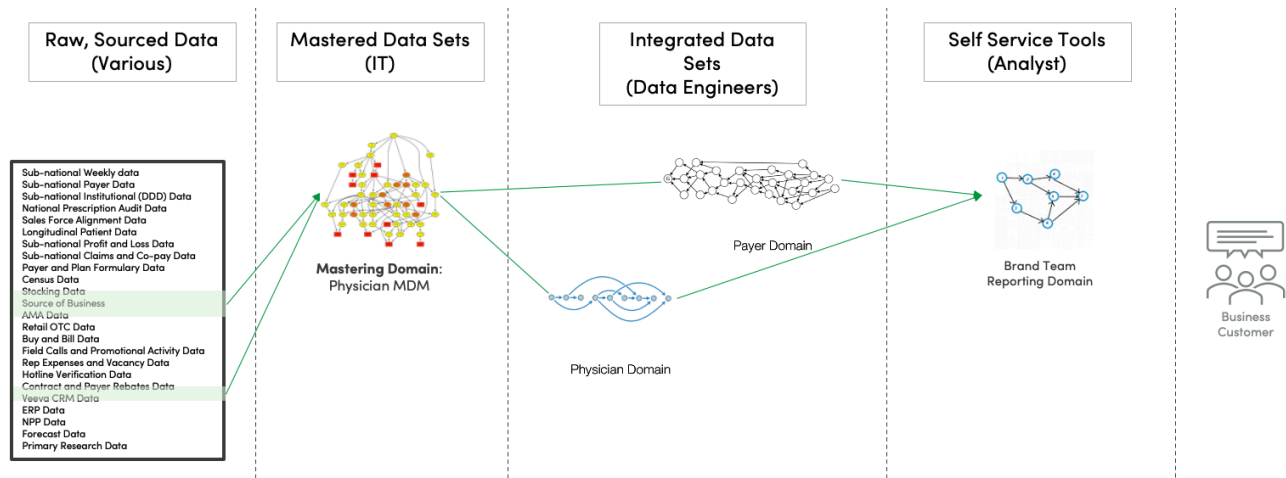


The second set of domains are the integrated data sets created by data engineers in the form of data warehouses or analytic data marts. There are facts and dimensions, along with multiple tables, used to answer business questions that come up. In many cases, these are star schemas.

The third set of domains are cached data sets (e.g., tableau extract) or small data sets that self-service analysts can mix with the central data in Alteryx or other tools. Self-service data science teams may require their own segmentation models for building reports, views, and PowerPoints.

Each circle in the figure above could be its own data mesh domain. The separation of these data sets enables the teams to decouple their timing from each other.

In figure 12 below, we see that each domain has its own domain update processing (recipes or data analytics pipelines), represented by a **directed-acyclic graph (DAG)**. For each domain, one would want to know that a build was completed, that tests were applied and passed, and that data flowing through the system is correct.



There's a top-level DAG relationship between the raw data, arriving asynchronously, the mastered domain, the integrated data sets and the self-service domain. One challenge is that each domain team can choose a different toolset that complicates multi-level orchestration, testing and monitoring. A DataOps superstructure that supports connectors to data ecosystem tools addresses this issue.

Another challenge is how to manage ordered data dependencies. The American Medical Association (AMA) may update its data set of physicians which flows into the physician mastered domain. It then gets used by the physician and payer data warehouses which are eventually used by the self-service teams. We don't want the physician data set in the physician domain and the payer domain to drift apart or get "out of sync," which might happen if they are updated on different iteration cadences. While the goal of a data mesh is to empower teams through decentralization, the overall architecture has to consider the order-of-operations dependencies between the domains. If not intuitively obvious, this is incredibly hard and a major cause of unhappy users.

To manage complexity, the system requires inter-domain communication shown in the table below. A domain query provides information about builds, data, artifacts, and test results. Process linkage aids in multi-level DAG orchestration. Some designs perform process linkage with an event bus that perhaps

marks the completion of a DAG by putting an event on a Kafka queue and using a publish/subscribe model. Data linkage refers to the sharing of common tables or the output of data from one domain being fed into another. Finally, there is development linkage. Can development environments with all related domains be created easily? Can they be modified? Is there a seamless path to production?

The many ports or access points of inter-domain communication further illustrate why it's so helpful to support URL-based queries. It enables the data engineer to ask questions, parse responses and set up automated orchestrations, instead of maintaining checklists and manual procedures that someone must execute.

We see a DataOps superstructure like DataKitchen DataOps Automation playing a central supporting role in the successful implementation of a data mesh. DataOps excels at the type of workflow automation that is able to coordinate interdependent domains, manage order-of-operations issues and handle inter-domain communication. The next section will explore the DataOps-enabled data mesh in more depth.

Inter-Domain Communication	Question / Steps Asked
Domain Query	"When was the last time you were updated? Successful or failure? Warnings?" "Is the data or artifacts in your domain good?" "Can you prove it with some test results?"
Process Linkage	"Ok, you start. I am done." "Ok, you start. I am done AND here are a bunch of parameters you need to keep going."
Event Linkage	"Here is an event: e.g., processing completed, error, warnings, etc."
Data Linkage	"We share a common table (e.g., a dimension table) in our domain."
Development Linkage	"Can I re-create your domain in development?" "Can I see the code you used to create it?" "Can I modify that code in development?" "Is there a path to production?"

Table 2: Inter-Domain Communication

DataOps and the Data Mesh

DataOps focuses on automating data analytics workflows to enable rapid innovation with low error rates. It also engenders collaboration across complex sets of people, technology, and environments. DataOps produces clear measurement and monitoring of the end-to-end analytics pipelines starting with data sources. Whereas data mesh talks about architecture and team alignment, DataOps automates workflows that simplify data mesh development and operations. A data mesh implemented on a DataOps superstructure, like DataKitchen DataOps Automation, has a much greater chance of success. One can

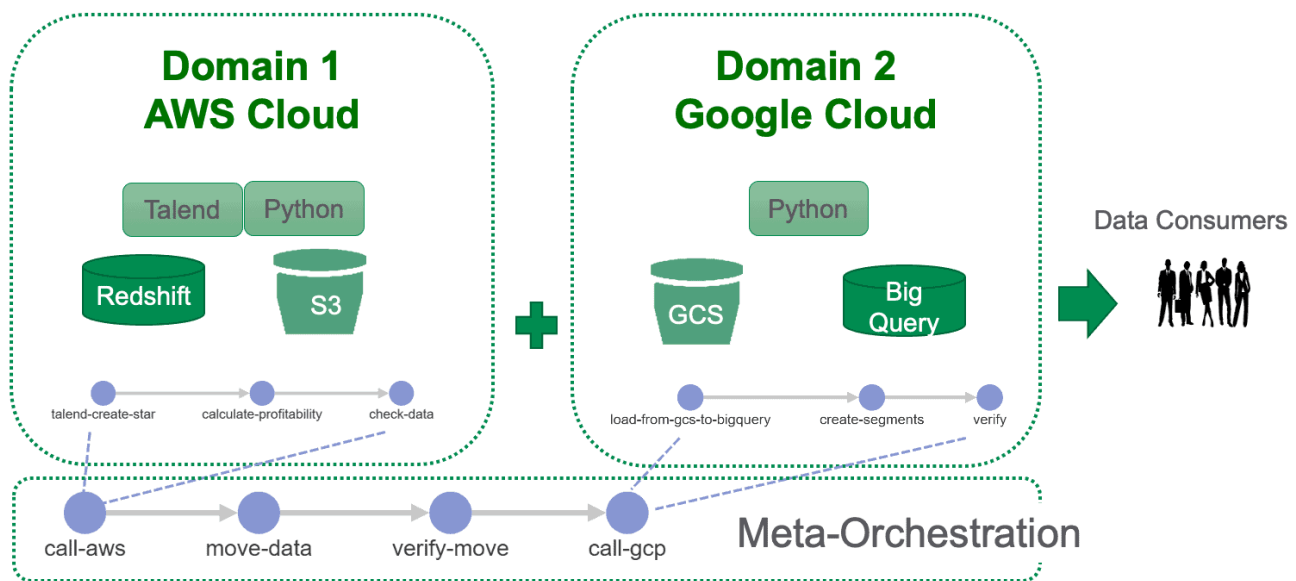
implement DataOps on its own without data mesh, but data mesh is a powerful organizing principle for architecture design and a great fit for DataOps-enabled organizations. **DataOps is the scaffolding and connective tissue** that helps you construct and connect a data mesh.

A discussion of DataOps moves the focus away from organization and domains and considers one of the most important questions facing data organizations - a question that almost never gets asked. "How do you build the factory that makes the machines?" The data factory takes inputs in the form of raw data and produces outputs in the form of charts, graphs and views. Before you can run your data factory, you have to build your data factory. Before you build your factory, you would do well to design mechanisms that create and manage your data factory.

Architecture, uptime, response time, key performance parameters - these are challenging problems and so they tend to take up all the oxygen in the room. Take a wider view. Architect your data factory so that your data scientists lead the industry in cycle time. Design your data analytics workflows with tests at every stage of processing so that errors are virtually zero in number. Doing so will give you the agility that your data organization needs in order to cope with new analytics requirements. Agile analytics will help your data teams realize the full benefits of an application and data architecture divided into domains.

Efficient workflows are an essential component of a successful data team initiative. One common problem is that code changes or new data sets may break existing code. Your domain DAG is ingesting data, and then transforming, modeling and visualizing it. It's hard enough to test within a single domain, but imagine testing in relation to other domains which use different teams and toolchains, managed in other locations. How do you allow a local change to a domain without sacrificing global governance and control? That's important to do, not only within a single domain but between a group of interdependent domains as well.

With a DataOps superstructure, like DataKitchen DataOps Automation, you are testing each step in your DAG on whatever infrastructure and tools the domain is using. DataKitchen supports an intelligent, test-informed, system-wide production orchestration (meta-orchestration) that spans toolchains. Unlike orchestration tools like Airflow, Azure Data Factory or Control-M, DataKitchen can natively connect to the complex chain of data engineering, science, analytics, self-service, governance and database tools, and meta-orchestrates a hierarchy of DAGs. Meta-orchestration in a heterogeneous tools world is a critical component to successfully rolling out a data mesh.



DataKitchen DataOps Automation natively provides URL access to nearly all the interfaces that are required for inter-domain communication.

Inter-Domain Communication	Question / Steps Asked	DataKitchen Support
Domain Query	"When was the last time you were updated?" "Successful or failure? Warnings?" "Is the data or artifacts in your domain good?" "Can you prove it with some test results?"	Yes
Process Linkage	"Ok, you start. I am done." "Ok, you start. I am done and here are a bunch of parameters you need to keep going."	Yes
Event Linkage	"Here is an event: e.g., processing completed, error, warnings, etc."	Yes
Data Linkage	"We share a common table (e.g., a dimension table) in our domain."	Link to 3rd Party Tools
Development Linkage	"Can I re-create your domain in development?" "Can I see the code you used to create it?" "Can I modify that code in development?" "Is there a path to production?"	Yes

Table 3: DataKitchen DataOps Automation support for inter-domain communication

The capability to execute domain queries comes from DataKitchen **order** runs. Process queries come from calling an order run or running a **recipe** (orchestration pipeline). Event linkage is natively supported and development linkage stems from **Kitchens** (on-demand development sandboxes). DataKitchen

supports data linkage by integrating with tools that access and store data - there are plenty of great ones.

DataKitchen groups recipes into components called **ingredients**. Ingredients are composable units that can enable domains to change independently. They can also be orchestrated together within Kitchens.

DataKitchen can help you address the important tasks of how to develop, deploy, and monitor analytics related to a domain.

- **Recipes** - Orchestrate the whole pipeline (i.e., ingest, process/curate, serve, etc.) inside of a domain. DataKitchen Recipes can serve as a master DAG as well as lower-level DAGs nested inside other DAGs.
- **Monitoring Tests** - Make sure the data (from suppliers and to customers) is trustworthy. This aids in diagnostics (i.e., detection and localization of an issue). In the DataKitchen context, monitoring and functional tests use the same code.
- **Variations** - Execute data pipelines with specific parameters. Deliver the correct data product for the consumer, because one size does not fit all. Variations enable the data team to create different versions of their domain to handle development, production, “canary” versions or any other change. Variations unlock a great degree of agility and enable people to be highly productive.
- **Kitchen Wizard** - Provide on demand infrastructure to the data teams to prevent delays. One concern related to domain teams is the potential duplication of effort with respect to horizontal infrastructure. DataKitchen can be used to automate shared services. For example, DataKitchen provides a mechanism to create **self-service, development sandboxes** so the individual teams do not have to create and support this capability.
- **Kitchens/Recipes/Functional Tests** - Iterate to support the “product-oriented” approach. Building the factory that creates analytics with minimal cycle time is a critical enabler for the customer focus that is essential for successful domain teams.

Conclusion

Data mesh is a powerful new paradigm which deals with the complexity in giant, monolithic data systems. As an organizing principle, it focuses on data, architecture and teams and less, the operational processes that are so important for agile, error-free analytics. As part of your data mesh strategy, DataOps assists with the process and workflow aspects of data mesh. DataOps automates shared services preventing duplication of effort among teams. DataOps also addresses some of the complexity associated with domain inter-dependencies and enables the data organization to strike the right balance between central control/governance and local domain independence.

SECTION 6

Running it day to day

1. The Five Organizational Sins of Data Teams
2. How Do I Get My Data Team More Efficient?
3. Webinar Summary: Driving Data Analytic Team Excellence Through Agility, Efficiency, and Aphorisms

The Five Organizational Sins of Data Teams

Read the original article at: <https://datakitchen.io/blog/the-five-organizational-sins-of-data-teams/>



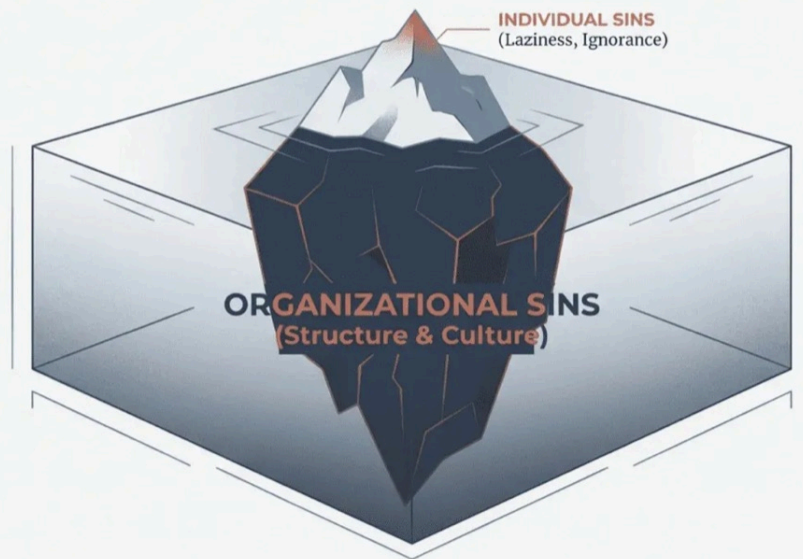
In the first part of this series, we explored the [Seven Deadly Sins of Data Quality](#), focusing on the individual behaviors that sabotage data teams: shame, denial, avoidance, passivity, laziness, gluttony, and ignorance. But here is the uncomfortable truth that nobody wants to talk about: even if every person on your data team somehow conquered their personal demons tomorrow, you would still have a data quality problem. The issue is not just about people. It is about how organizations structure their teams, define success, and approach the very nature of data work itself.

IT'S NOT YOUR PEOPLE. IT'S YOUR STRUCTURE.

The Uncomfortable Truth: Even if every person on your team conquered their personal demons today, you would still have a data quality problem.

The Reality: We blame data quality on individual behaviors, but the root causes are baked into the culture, incentives, and operating models.

The Diagnosis: We will examine five systemic pathologies that turn talented individuals into ineffective teams.



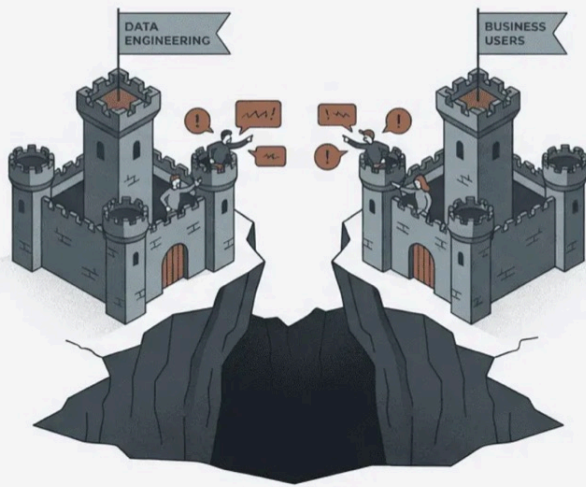
Organizational sins are more challenging to diagnose and even more complex to fix because they are baked into the culture, the incentive structures, and the operating models that everyone takes for granted. These are the systemic dysfunctions that turn talented individuals into ineffective teams and promising initiatives into expensive disappointments. In this follow-up post, we examine five organizational sins that plague data teams: low relational coordination, blame culture, data blindness, project thinking, and lack of process curation. Recognizing these patterns is the first step toward building data organizations that actually work.

Sin One: Low Relational Coordination

Data organizations face unique challenges in fostering positive interaction and communication, as tools and workflows often promote isolation. Groups that should be partners working toward shared objectives often behave more like warring tribes competing for resources and pointing fingers when things go wrong. As explored in [Warring Tribes into Winning Teams](#), geographical, cultural, and language barriers compound with technology-driven silos to create what management scientists call low relational coordination. The data engineers, analysts, scientists, and business users who need to collaborate seamlessly instead operate in isolated bubbles, communicating through emails and handoffs rather than a genuine partnership.

SIN #1: LOW RELATIONAL COORDINATION

Teams behaving like warring tribes rather than partners.



The Trap: Technology-driven silos and geographical barriers create isolation. Groups that should be partners operate as competitors.

The Behavior: Communication happens through ticket handoffs and “throwing requirements over the wall.”

The Result: “Us vs. Them” mentality. Engineers build pipelines without understanding context; business users create shadow IT.

The consequences ripple through every aspect of data operations. Analytics cycle times stretch as teams wait for responses from groups they barely know. Data quality suffers because no one has visibility into the entire pipeline. Governance becomes a bureaucratic exercise rather than a shared commitment. Employee retention plummets as talented people tire of fighting the same battles over and over. The solution requires more than better communication tools. Organizations must restructure their workflows to build strong role relationships and coordination that enable high performance. DataOps provides a framework for doing exactly that, using automation and process design to facilitate rather than hinder the communication between groups.

Sin Two: Blame Culture

In the world of data and analytics, one skill stands timeless and universal: the art of blaming someone else when things go sideways. As satirized in [The Art of Data Buck-Passing 101](#), data teams have elevated finger-pointing to a dark art form. When data issues surface, development teams blame QA. QA blames operations. Operations blames the data providers. Everyone invokes the sacred incantation of “garbage in, garbage out” while carefully positioning themselves outside the blast radius. The [Spiderman pointing meme](#) has become the unofficial mascot of data team retrospectives.

SIN #2: BLAME CULTURE

The art of data buck-passing.



The Trap: When data breaks, the organization asks “Who did this?” rather than “How do we fix this?”

The Behavior: “Garbage in, garbage out” becomes a shield to deflect responsibility. Dev blames QA. QA blames Ops. Ops blames Source.

The Symptom: The “Shame Game”—engineers avoid looking for errors because finding one means owning the blame.

This blame culture is not just demoralizing; it is organizationally corrosive. When people fear being blamed for problems, they stop looking for problems. They push untested code to production because once it is deployed, someone else becomes responsible. They craft emails that document decisions in ways designed to deflect future accountability rather than facilitate current collaboration. The energy that should flow into fixing root causes instead flows into constructing elaborate defenses. Breaking this cycle requires shared visibility into data pipelines, clear ownership models, and a cultural shift from “who broke it” to “how do we prevent it from breaking again.” Until organizations address the blame game directly, they will continue playing it.

Sin Three: Data Blinders

Most data teams are data-centric rather than process-centric, and this distinction has profound implications for their effectiveness. As described in [Are You Process-Centric or Data-Centric?](#), data-centric teams perceive complexity as directly proportional to the number of tables they manage. Their databases feel like mysterious, daunting black boxes full of tables updated at seemingly random frequencies by processes that are unknown or uncontrolled. Data engineers in these environments frequently respond to quality issues without direct control over the processes that cause the errors. They are held responsible for data problems but cannot actually fix or influence the pipelines that create them.

SIN #3: DATA BLINDNESS

Drowning in tables, starving for control.



- **The Trap:** Being Data-Centric instead of Process-Centric.
- **The Behavior:** Measuring complexity by the number of tables managed. Treating databases as mysterious "black boxes" that update randomly.
- **The Reality:** You are held responsible for errors in pipelines you cannot see or control. You fix the data, but not the process that broke it.

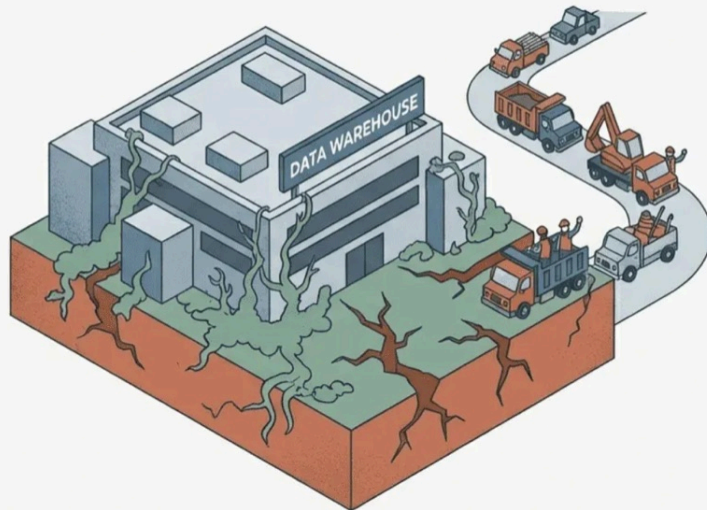
Process-centric teams operate completely differently. They focus on orchestrating and automating workflows, recognizing that robust, well-managed data processing pipelines inevitably yield reliable, high-quality data. Rather than focusing on individual tables, these teams allocate resources to ensure each pipeline is transparent, thoroughly tested, and easily deployable. They treat their pipelines as cattle, not pets, enabling them to quickly diagnose and rectify issues at the process level rather than getting lost in individual data table anomalies. The shift from data-centric to process-centric thinking is achievable, but it requires leadership commitment to unified orchestration, process transparency, and rigorous adherence to DataOps principles.

Sin Four: Project, Not Product Focus

When data team leaders hear “data products,” they immediately think of the stuff they produce: dashboards, datasets, models, and warehouses. But as argued in [You’re Thinking About Data Products All Wrong](#), focusing on the “what” completely misses the massive shift that data products really represent. The truth is that data products are not about what you create but how you build, maintain, and continually improve your data assets to deliver value to customers. When organizations think about data products merely as deliverables, they fall into the same trap that has plagued data teams for decades: the project mindset.

SIN #4: PROJECT, NOT PRODUCT FOCUS

The “One-Off” Trap.



- **The Trap:** Treating data work as finite projects with a “done” date.
- **The Behavior:** Success is defined by delivery on time/budget, not by long-term value or reliability.
- **The Flaw:** Data is never “done.” It evolves, drifts, and breaks. The “Ostriches” bury their heads after deployment, ignoring downstream decay.

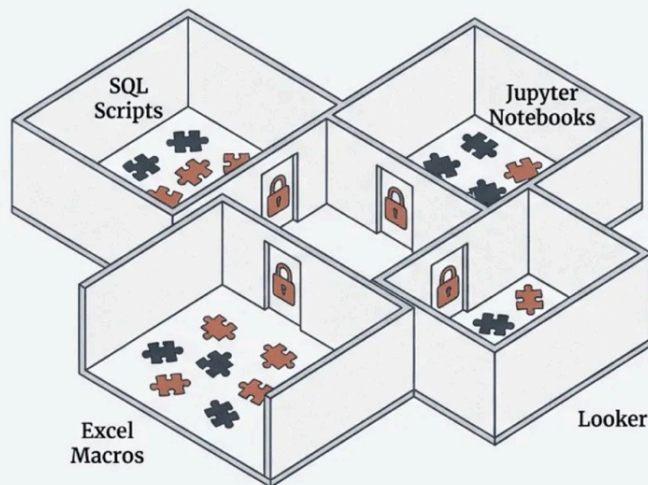
In the project world, workflows progress through stages in sequence. Teams are assigned to specific projects, creating silos and handoffs that slow delivery and degrade quality. Success is measured by whether the project was delivered on time and within budget, not by whether users actually found it valuable. The product approach transforms this dynamic entirely. Instead of significant, risky releases, teams deliver minor, incremental improvements continuously. Work is brought to stable teams who own their products end-to-end. Most importantly, the approach recognizes that data products are never “done.” This continuous improvement mindset yields measurable benefits: faster releases, increased team learning, better insights, and ultimately faster delivery of value from data investments.

Sin Five: Lack of Process Curation

There is a quiet crisis happening in data organizations everywhere. It occurs when brilliant analysts independently solve the same problem in three different ways, when data scientists struggle to reproduce each other’s work, and when a single “simple” metric like Monthly Recurring Revenue has five competing definitions scattered across the organization. As detailed in [Process Guardianship: The Most Valuable Data Engineering Work You’re Probably Not Doing](#), the absence of deliberate process curation means that business logic lives everywhere and nowhere. Metrics, models, calculations, and code are scattered across SQL scripts, Jupyter notebooks, Looker reports, Python models, and Excel macros, with no one taking responsibility for consolidating them into a unified, tested, production-grade system.

SIN #5: LACK OF PROCESS CURATION

Business logic: everywhere and nowhere.



- **The Trap:** Important logic is scattered across hidden corners and tribal knowledge.
- **The Behavior:** A simple metric like “Monthly Recurring Revenue” has five different definitions depending on who you ask (and which Excel sheet they use).
- **The Reality:** No deliberate consolidation or guardianship of the process.

The costs compound quickly. Duplicate work becomes the norm as teams waste countless hours reinventing the wheel because they do not know what already exists or cannot trust it enough to reuse. Multiple overlapping metrics create confusion and erode stakeholder trust. When leadership asks how many customers are in the high-value segment, they get three different answers. Embarrassing data errors slip through because corrections identified by one team are not shared with the rest of the enterprise. Data engineers have a unique opportunity to address this crisis through what might be called process guardianship: actively consolidating business logic, extracting it from hidden corners and tribal knowledge, and encoding it in a centralized, version-controlled, production-ready system. This work may seem unglamorous, but it is often the most valuable contribution a data team can make.

Moving From Organizational Sin to Organizational Strength

These five organizational sins share a common thread: they all represent failures of systems and structures rather than failures of individuals. You cannot shame individual data engineers into fixing low relational coordination. You cannot train analysts out of blame culture while the incentive structures reward blame. You cannot wish your way from data-centric to process-centric thinking without fundamentally changing how work is organized and measured. This is precisely why DataOps exists. DataOps is not just DevOps for data. It is a comprehensive methodology that applies Agile development, DevOps, and lean manufacturing practices to data analytics, enabling organizations to accelerate the development of new analytics, deploy with confidence, and reduce data errors to virtually zero.

The [DataOps Cookbook](#) provides a practical roadmap for implementing these principles. Now in its third edition, the Cookbook explains how DataOps draws on lessons from the software and manufacturing industries to transform data chaos into a single, smooth, efficient process that delivers fast, error-free insight. It covers everything from understanding Data Journeys and implementing observability to building

the automated testing and unified orchestration that process-centric teams require. The methodology addresses each of the organizational sins we have discussed: it breaks down silos through shared visibility and collaboration frameworks, eliminates blame culture by making pipelines transparent and accountable, shifts teams from data-centric to process-centric thinking, enables product mindsets through continuous delivery practices, and provides the foundation for true process guardianship.

THE REVOLUTION STARTS WITH YOU

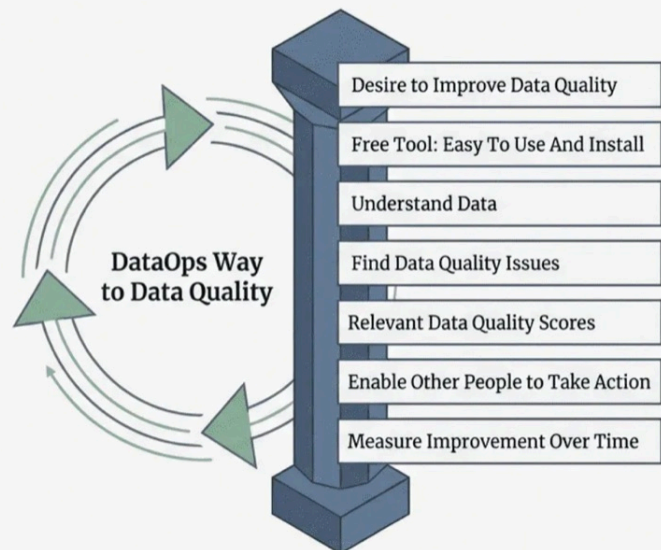
Don't wait for a culture shift.
Create one.

Step 1: Download DataOps TestGen
(It's Free & Open Source).

Step 2: Generate your first quality
report in minutes.

Step 3: Turn on the lights and see what
you've been missing.

<https://info.datakitchen.io/testgen>

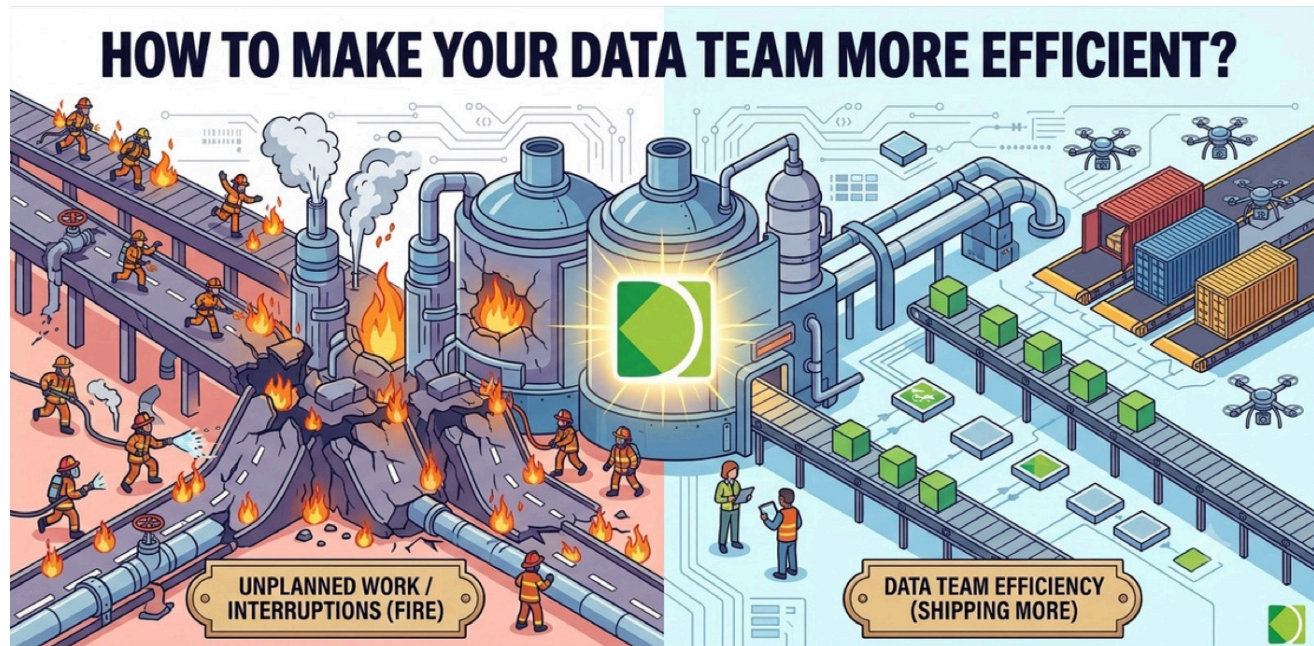


DataKitchen: The DataOps Experts.

The journey from organizational dysfunction to data team excellence is not easy, but it is achievable. Thousands of data professionals have already formed study groups around the DataOps Cookbook and are beginning to transform their organizations. The key insight is that you do not need to fix everything at once. Start by focusing on understanding and observing the journey that data takes through your production environment. This monitoring process identifies data errors, tool problems, and timing issues, enabling a quick win for your DataOps implementation by driving immediate improvements. Lowering production errors increases the reliability of your data and gives your team more time to focus on automation. From there, you can systematically address each organizational sin, building the collaborative, process-centric, product-oriented data organization that actually delivers on the promise of data-driven decision making. Download the [DataOps Cookbook](#) and start your team's transformation today.

How Do I Get My Data Team More Efficient?

Read the original article at: <https://datakitchen.io/blog/how-do-i-get-my-data-team-more-efficient/>



Your Monday starts with a Slack message. Somebody in finance says the numbers look wrong. Three people drop what they were doing. By four in the afternoon, you find it: an upstream system added a column and changed a type, and the load has been quietly dropping rows since Thursday. You fix it, you write the apology, you go home.

Nothing on that Monday was in the sprint. That's the whole problem.

When someone asks how to make a data team more efficient, they usually mean: how do we ship more? Typing faster won't do it. A large share of the week goes to **unplanned work**, and shipping more means reducing that number.

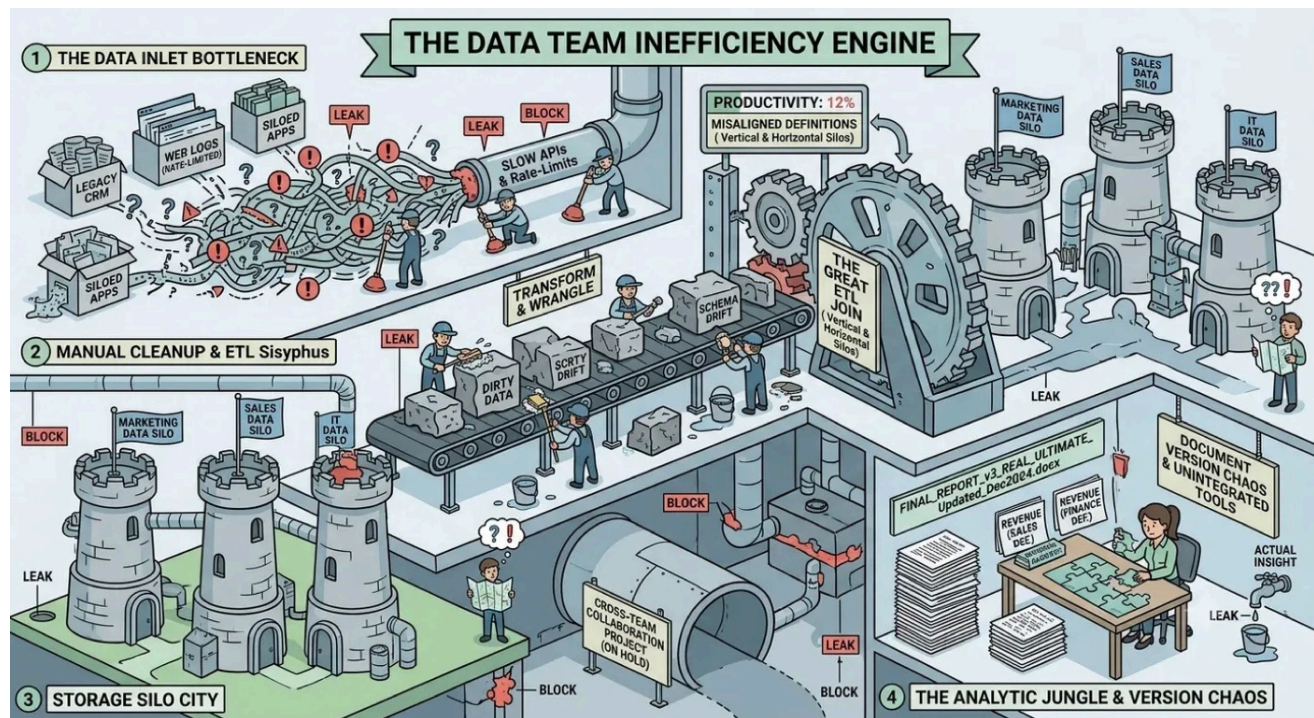
What the research actually says

You've seen the stats on bad data. The time ones hold up better than the money ones, so start there.

The most useful one comes from a 2022 survey of more than 300 data professionals, fielded by Wakefield Research and **sponsored by Monte Carlo**. Respondents said 40 percent of their time goes to evaluating or checking data quality. Three quarters of them need four hours or more just to detect an incident, and about half need another nine hours to resolve it once they've found it. Call it 13 hours per incident, between the break and the fix, before anyone corrects a single downstream report. More than half also said incidents were going up, not down.

That is the time cost. There is a human one underneath it, and the best evidence I know of is ours. In 2021 we surveyed 600 data engineers with data.world, also fielded by Wakefield. 97 percent reported burnout in their day-to-day work. The causes they named were not exotic: too much time finding and fixing errors, manual repetitive work on data prep and pipelines, and a pace of incoming requests nobody could stay ahead of. The same three things this post is about.

When the subject comes up, somebody usually reaches for governance. That survey is unkind on the point. 69 percent of those engineers said their own company's data governance policies made their day-to-day job harder. Governance gets sold as the fix and lands as friction.



The tactics below assume a small or mid-size data team, where you know everyone who is breaking things and you can change something without a review board. Inside a large enterprise the diagnosis holds and the tactics don't, because getting a check into production there is a procurement exercise, not an afternoon. Three routes follow, and most teams need more than one.

First, get your own number

The money numbers are softer than the time ones, and you should know why before you put them on a slide. Gartner's widely quoted \$12.9 million average annual cost of poor data quality comes from its 2020 data quality Magic Quadrant, where 154 reference customers of data quality vendors were asked to estimate their own losses. Self-reported guesses from large enterprises who were already buying the software. Fine as an order of magnitude, useless as a benchmark for your team. People cite IBM's \$3.1 trillion figure from 2016 even more often, and it has never carried a published methodology. MIT Sloan's estimate that companies lose 15 to 25 percent of revenue to bad data is in the same family.

Use those to establish that the problem is real and expensive. Don't take them to your CFO as evidence about your own team. Take your own number instead, and you can have it in a month.

Before you pick a route, measure what's actually happening. Not with a timesheet. Your engineers already resent the tracking they have, and asking them to log every interruption will get you a spreadsheet with four entries.

Use what you already collect. Tag every ticket that came from something breaking rather than something being planned, then read the ratio at the end of the sprint. Or count incidents per week and multiply by an honest estimate of what one costs end to end. Or spend an hour scrolling back through the channel where people report broken dashboards and reconstruct last month. Any of these gets you a number you can defend in a room, and none of them needs a new process.

Then track detection lag: from the moment bad data lands to the moment a human says something. You probably know this one already. If the answer is "a day or two, unless it's the exec dashboard, then 20 minutes," you have a detection problem, not a data problem. You are **eyeballing row counts and praying** whether you call it that or not.

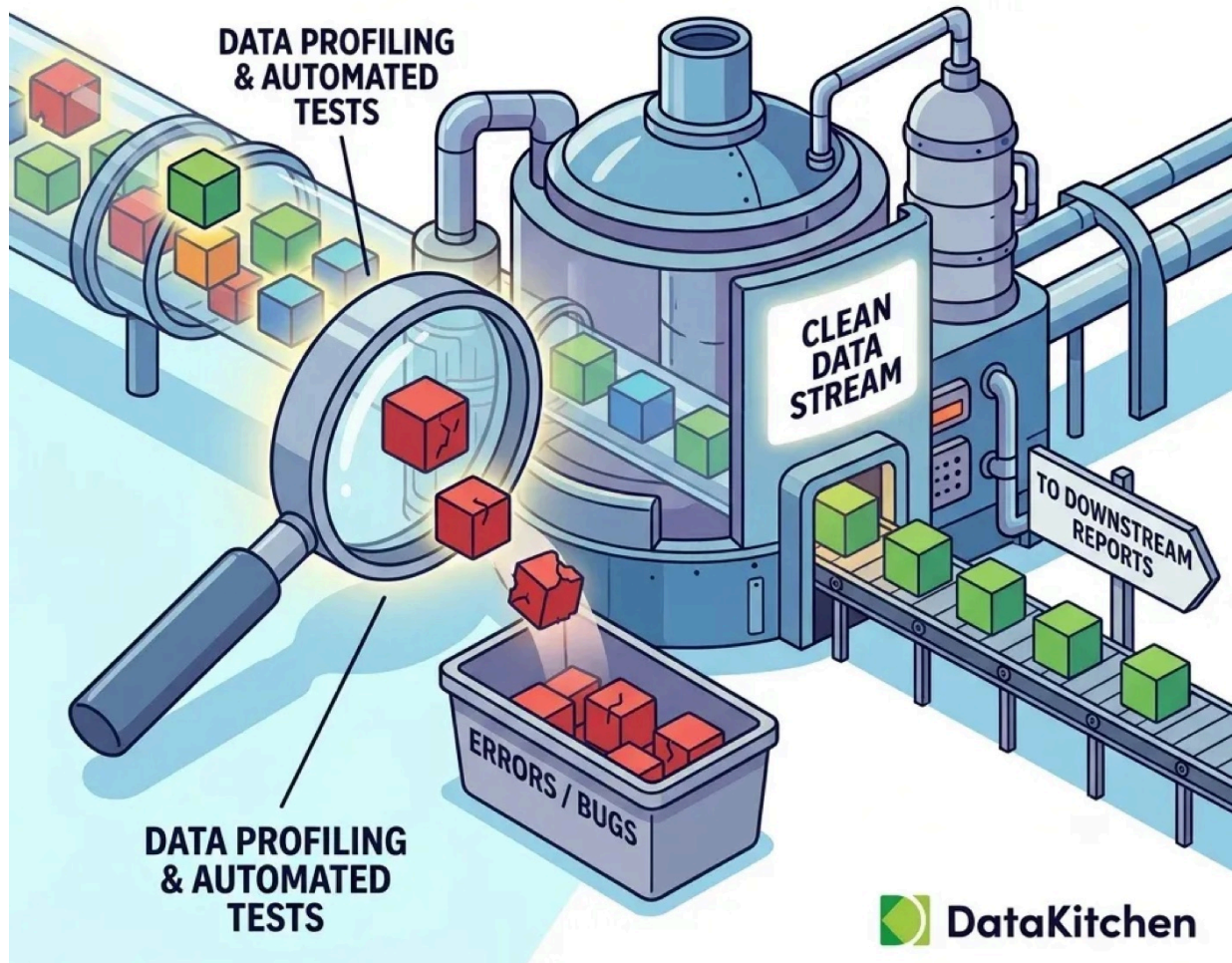
Those two numbers tell you which route to take, and they give you a before, which you'll want when someone asks in six months whether any of this worked.

Route one: stop the errors

This is the highest return per dollar, and almost nobody starts here, because it isn't exciting.

Production errors are a time problem before they are a quality problem. Every one of them buys you an investigation, a fix, a re-run, a set of downstream corrections, and a conversation with whoever built a report on the broken table. The fix takes 20 minutes. Everything around it takes a day.

ROUTE ONE: STOP THE ERRORS.



The reason errors cost so much is that you find out about them from a person. Somebody who trusted your table enough to build on it is now your monitoring system. An expensive alerting layer, and it costs you a little credibility every time it fires.

Be precise about what tests fix, because many vendors are not. Most of your breakage starts upstream, in systems you don't own, run by people who don't know you exist. Tests don't prevent that. The Salesforce admin will still add a field on a Tuesday without telling anyone. What tests change is when you find out and how far it spread. Monday at 9am from finance becomes Tuesday at 6:14am from a check that ran before anything downstream read the table. The claim stops there. It's enough.

The counter is a test that runs before humans read the data. The objection is always that nobody has time to write tests for 4,000 tables, which is true and also the wrong frame, and we [got roasted on Reddit for asking about it](#). You don't write them. You [profile what the data actually does and generate tests from](#)

that: column ranges, null rates, cardinality, freshness, referential integrity, distribution shifts. Then you keep the ones that make sense. An afternoon, not a project.

Start where a wrong number reaches someone who can escalate it. Usually a couple dozen tables, not 4,000, which is what **test coverage as a measure** actually means, and it's a better filter than picking by volume or by which pipeline annoys you most.

Then tune, because this is where generated tests usually die. Profile-based checks will flag legitimate variation. Somebody mutes the channel, and six weeks later the tests are decoration. Split them from the start. Hard constraints you know are true, like a key being unique or an amount never going negative, can fail loudly on day one. Statistical checks like distribution shifts and volume ranges run in warn-only mode for a couple of weeks while you learn what normal looks like. Promote them once they stop crying wolf.

Wire the ones you trust into the path your changes already take. On a normal stack, that means the tests run in the pull request, and the same checks run again against production data right after the load, before anything downstream reads the table. That is what **shifting left and shifting down** means in practice.

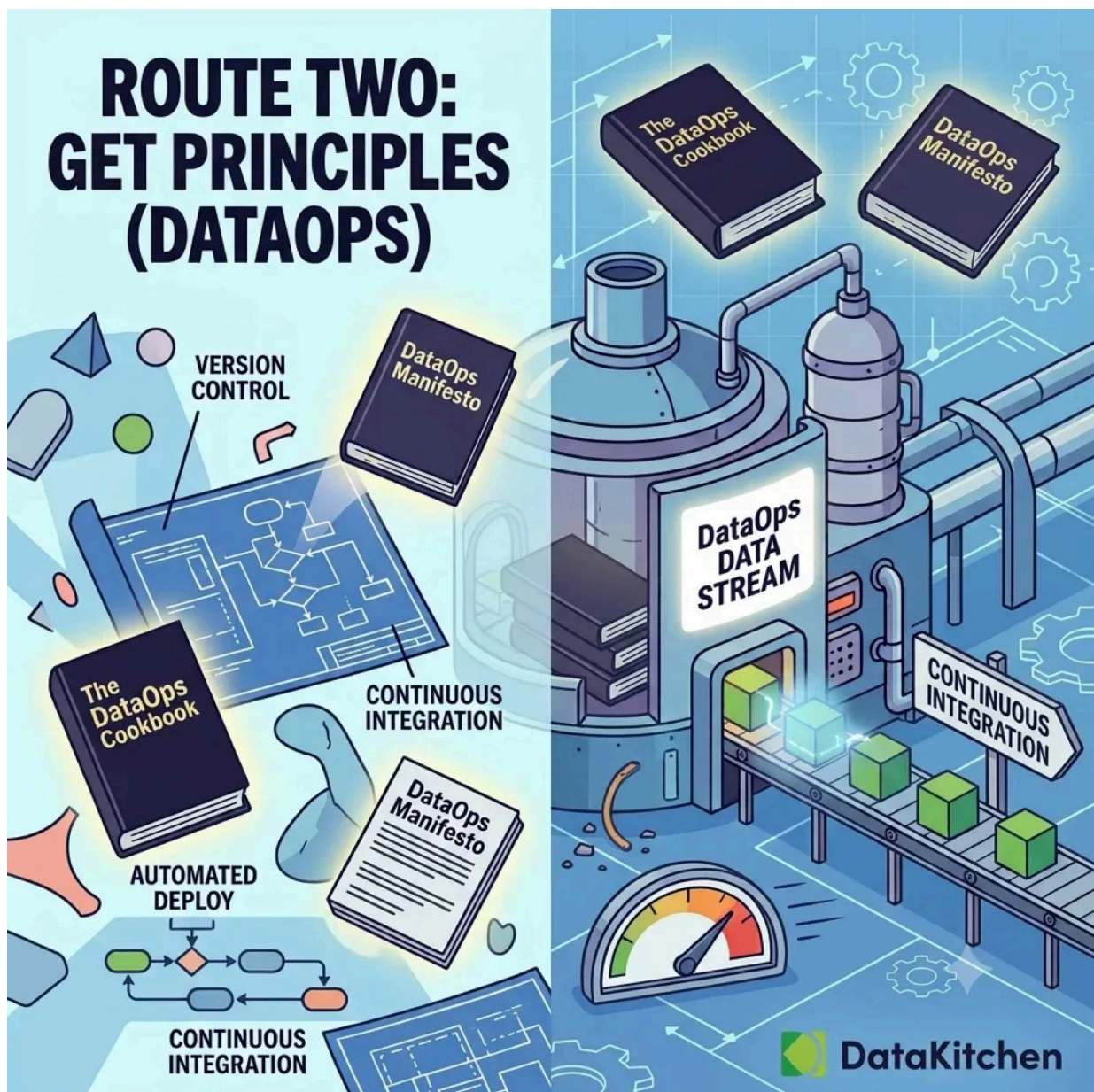
Decide in advance what a failure does, because halting is not free. Stop the pipeline and finance opens an empty dashboard instead of a wrong one, and now you're explaining that instead. Your rule: halt when the data would be silently wrong, warn when it would be obviously incomplete. A wrong number gets used. A missing one gets questioned.

This route works on the system you already have. No migration, no reorg, nothing to procure. It's also the only one of the three you can start without asking anyone for permission, which matters more than it should.

Route two: steal the principles

The second route costs the least money and the most patience. Start it yourself on Monday, for free.

Hold a **blameless postmortem** on every data incident. 15 minutes, written down: what broke and why, no names attached to blame. Keep the list somewhere shared. Read it once a month. After a quarter, the things eating your week will have named themselves, and you'll be holding the evidence you need to ask for anything bigger.



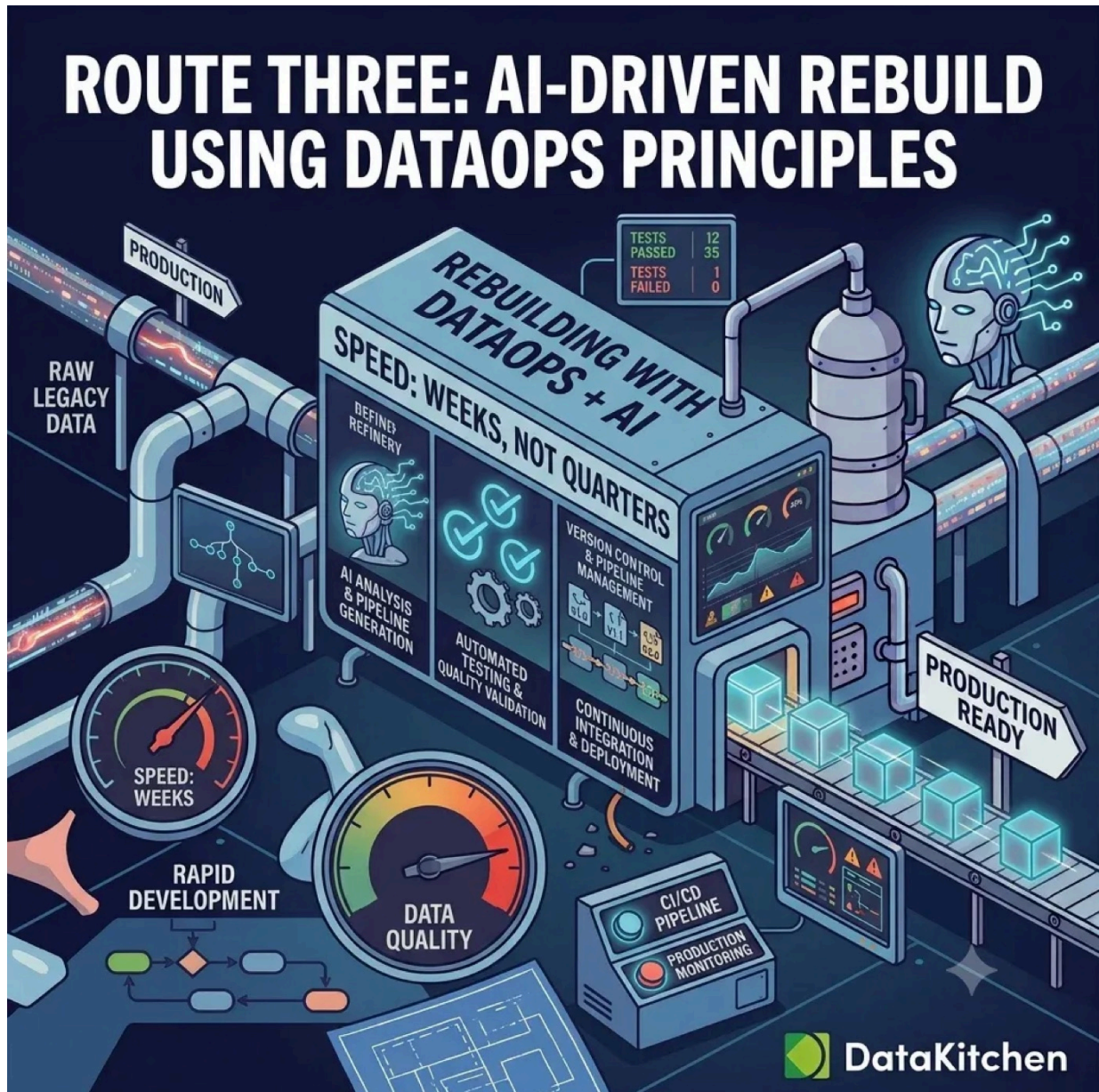
Then bring in people who have made data teams faster before, and take their principles rather than their labor. This works because **much of what slows you down is not technical**. It's that every change requires a human to verify it manually. It's that nobody agrees on which tables matter, so either everything gets watched or nothing does. It's that one person knows why that filter is there, so when they're on vacation, changes stop or break. That last one has a name, **process guardianship**, and it is usually nobody's job. No tool fixes those. A different way of working does.

The ideas are not secret. Software teams and manufacturing plants solved most of this decades ago with version control, automated testing, continuous integration, and statistical process control. Read the foundational **DataOps** material before you pay anyone, because then you can tell whether the consultant across the table knows the practice or is selling you a maturity model. If you want that material in an order, **Where to Actually Start with DataOps in 2026** lays out what to hand your team and what to do first.

The honest warning on this route: principles only move the number if **leadership changes what it rewards**. If your team is still measured by tickets closed, you'll end up with the best firefighters in the industry and no fire code. Advice can't fix an incentive.

Route three: rebuild it, if it needs rebuilding

The most expensive route, and the one to be honest with yourself about. Plenty of the pipelines you hate are fine underneath and just have no tests. Rebuild when the architecture is the problem, not when the last six months were painful.



The classic rewrite failed for a predictable reason. It took 12 months, the old pipelines kept breaking the whole time, and at month nine the sponsor changed and the project got shelved. Everyone reading this has seen that happen at least once.

AI changed the arithmetic, though less than the pitch decks claim. What collapsed is the mechanical work. A transform that took an engineer a day takes a couple of hours now, and that is the part we **can show from our own work**. It works best when the pipelines are **functional and idempotent**, because that is what lets AI reason about them and change them safely. What did not collapse is the rest of it: agreeing what the output should be, running old and new side by side until the numbers match, and cutting over without breaking the reports people already depend on. That still sets the calendar. So the rewrite is a smaller bet than it was three years ago, which is worth repricing if that is the last time you looked at it. Anyone quoting you a fixed number of weeks has not seen your pipelines.

Two things determine whether it works.

The first is who runs it afterward. A rebuild that ends with an outside team holding the keys has only moved your bottleneck somewhere with a contract attached. The handover is the deliverable, not the code.

The second is tests, which is why this route is third and not first. AI will generate a pipeline that looks right, runs clean, and is **quietly wrong in a way nobody notices for a month**. Without tests, you have built a much faster way to produce errors you can't see. If you're going to rebuild with AI, do route one first, or at a minimum in parallel. Otherwise you're buying speed and paying for it in Mondays.

The order

Be realistic about which of these you can actually start. Route one you can begin this week on your own authority. Routes two and three need a sponsor, because you don't set what your organization rewards and you can't authorize a rebuild. So do route one first. The number it produces is how you get the sponsor.

Most teams do this in reverse. They rebuild first because it feels like progress, hire a consultant when the rebuild stalls, and get to testing in year three when someone finally adds up the hours.

What next Monday looks like

The load runs at 6:14. A test fails on row count and the pipeline stops before anything downstream reads it. Somebody on your team gets a message at 6:20 with the table, the test, and the column that changed. They fix it before finance opens the dashboard.

Nobody outside the team ever knows it happened. That's what efficient looks like. It's boring, and boring is the point.

Webinar Summary: Driving Data Analytic Team Excellence Through Agility, Efficiency, and Aphorisms

Read the original article at: <https://datakitchen.io/blog/webinar-summary-driving-data-analytic-team-excellence-through-agility-efficiency-and-aphorisms/>



In the webinar “Driving Data Analytic Team Excellence Through Agility, Efficiency, and Aphorisms,” James Royster, Vice President of Commercial Operations, Insights, and Analytics at Karuna Therapeutics, shared his insights on leading efficient and effective data analytics teams. He drew from his twenty-five years of experience in business analytics, pharmaceutical brand launch strategy, and project management. James emphasized the importance of delivering value quickly, even if the solution is only 70% complete, rather than aiming for 100% completeness at the cost of significant delays. He encouraged attendees to adopt a proactive approach to problem-solving, focusing on preventing issues from arising rather than merely solving them as they occur. He also highlighted the importance of agility and adaptability in data analytics. Using humor and wisdom, James shared his experiences with various data professionals, from data engineers and data scientists to analysts. He stressed the importance of teamwork, comparing data analytics to a sports team that must work together to succeed.

Summary

James started by reminiscing about his early days as an analyst, when he spent 80% of his time on data cleaning and organizing, with very little time left for actual analytics. He then noted that the situation has reversed, with more time spent on analytics due to improved techniques, emerging technologies, and better processes.

James emphasized the importance of creating value for the customer and mentioned his Six Sigma lean training and the teachings of Deming and Toyota as key inspirations. He shared a story from a talk at a large consulting company in India, where he was asked who his competition was. After pondering the question, he answered “Amazon,” explaining that people’s expectations are shaped by their life experiences. The speed and convenience of services like Amazon have raised the bar for all industries, including data analytics.

**“I am competing
with Amazon”
&
“Unicorns exist”**



James stressed the need for data analytics teams to match customers’ expectations and provide insights quickly. He used the example of an ATM transaction, where data is readily available and updated in real-time on a customer’s mobile device. He argued that delivering value fast is achievable with the right mentality and team structure and that data engineers should continually strive to improve in this regard.

It is essential to recognize the evolution of the field and the changing expectations of data consumers. Emphasizing value, embracing agility and efficiency, and challenging teams to deliver high-value insights quickly are crucial factors for driving excellence in data analytics.

James also discussed the mindset of treating analytics teams like firefighters, responsive and agile when facing challenges. This approach emphasizes the importance of the first few months of launching a product in the pharmaceutical industry and how a strong launch can set a product’s performance trajectory. The concept of “commander’s intent” is mentioned as a way to enable teams to adapt and achieve their objectives even when plans don’t work out as expected.

The conversation then moved to the importance of logistics and data quality in analytics, particularly in the pharmaceutical industry. James highlighted the need for a **reliable data chain** to ensure the end analyst can focus on delivering value. Iterating and quickly addressing customer questions or concerns

improves outputs and conversations. A system with strong logistics is key to success in analytics and many other fields.

James emphasized the importance of speed, agility, and automation for effective data analysis. He compared the process to Instagram highlight reels, where the best insights come from multiple takes and iterations. Analysts must be able to iterate and quickly change the data to achieve meaningful results. He stressed the importance of embracing velocity, automating operations, and integrating quality into the data analysis.

James discussed mindset change, focusing on agility and DataOps. He observed that many teams fear challenges and fire drills but can achieve better outcomes by embracing velocity and the desire to improve. He noted the importance of moving away from manual operations, aiming for automation whenever possible. He advocated automating any process that would be performed more than once, so teams could focus on the next big question instead of repeating work. James discussed the challenges data engineering teams face in larger organizations with siloed structures. He emphasized the importance of ensuring that team members feel connected to their end customers, understand the value of their work, and are enthusiastic about their contributions.

How To Succeed?



A Mindset Change to DataOps

From	To
Change Fear	Change Velocity
Manual Operations	Automated Operations
Hope For Quality	Integrated Quality
Hero Mentality	Repeatable Processes
Perfection	70% Right The First Time

“Change your mindset”

He argued that if team members are merely going through the motions, it is crucial for the organization to take a step back and reassess its approach. He highlighted the innate human desire to belong to something bigger. He emphasized the need to foster a sense of belonging in each individual within the data logistics chain, including data scientists, engineers, analysts, and management.

To achieve this, James stressed the importance of engaging team members and helping them understand and feel connected to the bigger mission. He concluded by emphasizing the importance of aligning everyone within the data logistics chain towards focusing on value delivery and being part of meaningful activities contributing to customer success and improvement.

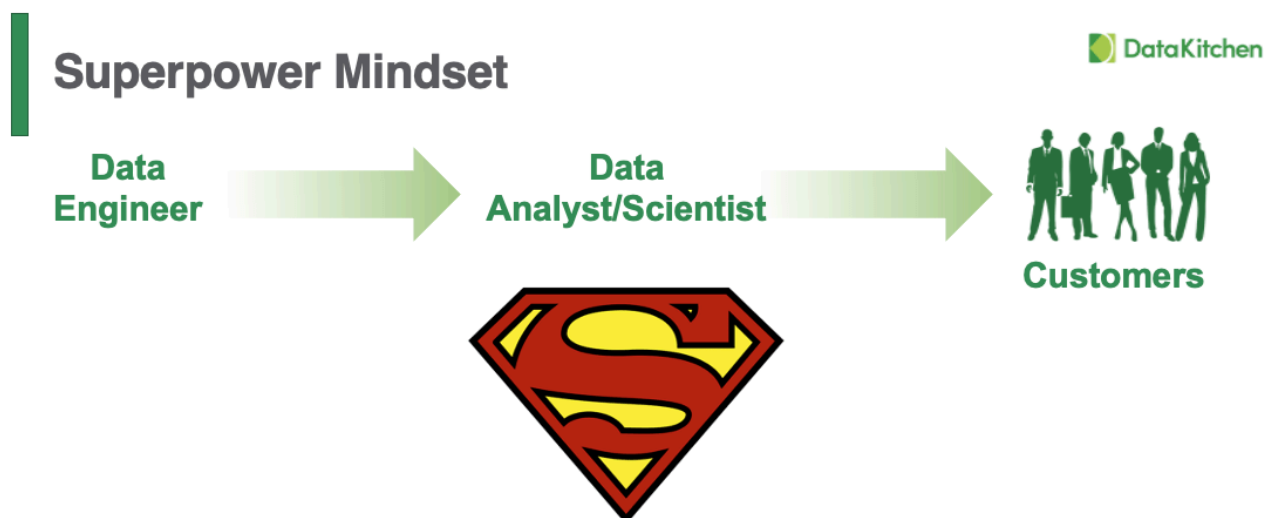
Testing, according to James, **is crucial in data analysis**. He shared an anecdote of a company that ran thousands of quality checks within seconds of receiving data, emphasizing the importance of ensuring the data's reliability. He touched on the importance of repeatable processes and standardization, noting that fixing problems should not be the main focus but rather preventing them from occurring in the first place.

Perfection is an unrealistic goal. Instead, he recommended aiming for 70-80% completion, then sharing the results with the customer for feedback. This approach allows for faster iteration and can help customers form their thinking. James also discussed the challenge of securing time with customers, suggesting that teams must ensure they provide value and engaging insights.

James emphasized that the key to effective data analysis is a combination of speed, agility, automation, and quality, focusing on providing value to customers across all levels, from sales representatives to CEOs.

James addressed the importance of cultivating a value-based mindset within a data engineering team. He emphasized embracing weaknesses and learning to improve rather than allowing external circumstances to hinder success. He shared that obstacles should be approached with well-developed business cases, noting that most people are supportive when presented with compelling reasons for change.

He underscored the relevance of a value delivery approach for data and analytics teams. While this approach can yield visible results, it leaves no room to hide when faced with challenges. Teams must confront their problems and prioritize those that offer the most leverage and long-term benefits.



***“The power to say... Sure we can do that...
We’ll have a first draft in a couple days “***

James introduced the concept of “hinge activities,” which he described as tasks that open doors to other opportunities. He stressed that the immediate needs of an organization must be addressed, but success in the future requires focusing on these hinge activities. This includes working on data quality testing and structuring data for easy access.

In conclusion, James emphasized that the key to effective data analysis is a combination of speed, agility, automation, and quality, focusing on providing value to customers across all levels. Cultivating a value-based mindset, prioritizing long-term benefits, and maintaining a strong customer connection are crucial factors for driving excellence in data analytics.

Read More

The following articles were not included in this edition of the book but cover related topics in depth. All are available free at datakitchen.io.

- [2024 Gartner Market Guide To DataOps](#)
- [DataKitchen Introduces DataOps Transformation Advisory Service](#)
- [Congratulations to the Karuna Team for their acquisition!](#)
- [Behind A Great Company, There is DataOps](#)
- [Reinvent Marketing Automation with DataKitchen DataOps Automation](#)
- [Accelerating Drug Discovery and Development with DataOps](#)
- [From Cattle to Clarity: Visualizing Thousands of Data Pipelines with Violin Charts](#)
- [How Three Small Pharma Companies Used DataKitchen to Achieve Commercial Launch Success and Skyrocket to \\$100 Billion in Acquisition Value](#)
- [Data Quality Circles: The Key to Elevating Data and Analytics Team Performance](#)