Nexusflow Inc.

PROCESS REFERENCE | VERSION 1.0

ETL and Dashboard Process Playbook

A reference for turning a one-person craft into a repeatable process that runs without you.



Discover | Extract | Transform | Gates | Serve | Operate

PREPARED BY

Nexusflow Inc

nxflo.com

EDITION

Version 1.0

November 2026

Contents

Start Here: The Whole Process on Two Pages . . . 3
1. Purpose and How to Use This Document . . . 5
2. Why Process Beats Heroics . . . 5
3. Input, Transform, Output: The Fundamentals . . . 5
4. Think Backwards: Output First, Then Input . . . 6
5. Worked Example: The FP&A Runway Dashboard . . . 8
6. The Pipeline at a Glance . . . 11
7. Roles and Ownership . . . 12
8. Stage One: Discover and Define . . . 12
9. Stage Two: Extract . . . 13
10. Stage Three: Transform . . . 13
11. Using AI to Build the Transform . . . 15
12. Stage Four: Quality Gates . . . 16
13. Test and Iterate . . . 17
14. Infrastructure and Cost Estimation . . . 18
15. Stage Five: Load, Serve, and Visualize . . . 19
16. Designing for the User . . . 20
17. Operating the System . . . 21
18. Change Control . . . 22
19. Security and Governance . . . 22
20. Handover: Designing Yourself Out . . . 22
21. Staying Valuable When the Process Runs Without You . . . 23
22. Metrics for the Process Itself . . . 24
In Closing: Focus on the Business . . . 25
Appendix A: Project Kickoff Checklist . . . 26
Appendix B: Runbook Template . . . 26
Appendix C: Go-Live Checklist . . . 27
Appendix D: Glossary . . . 27
Appendix E: Output Requirement Sheet . . . 28
Appendix F: Source Mapping Worksheet . . . 28

Start Here: The Whole Process on Two Pages

Every analytic dashboard, in any system, does the same three things: it takes something in, changes it, and puts something out. That is all ETL means.

Letter	Plain meaning	In the kitchen
E: Extract (input)	Get the data in	Buy the ingredients
T: Transform	Clean, combine, calculate	Cook
L: Load (output)	Put the answer where people use it	Serve the meal

The one idea: from your users you only need to pin down two things, the **output** they want to see and the **input** that can produce it. Everything in between is engineering, and AI can now help with much of it.

Just like cooking, we work in reverse. First decide what you want to eat, then go and find the ingredients. We make a wish about what we want to see in the report, and only then do we look for the data.

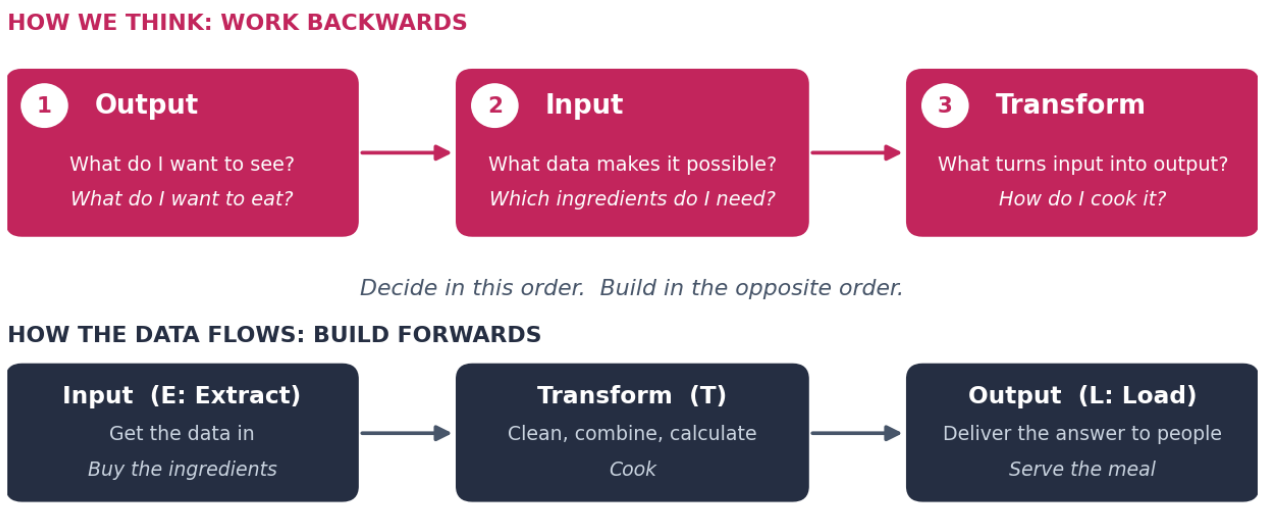


Figure 1. Think backwards, build forwards: the output decides the input, and the data is then built in its natural order.

A real example, the FP&A runway report. The finance team makes three wishes.

- **Runway:** from the monthly cash in, cash out, and budget, how many months of cash are left?
- **What-if:** if we increase the budget or cut expenses, how does the runway change? The user should be able to try it and see the answer at once.
- **Mid-year review:** how much of this year's budget has been spent, what is the projection for the rest of the year, how are the projects doing, and what should we adjust?

One more requirement hides in plain sight: how fresh the data must be. Real time, hourly, daily, or monthly? It sounds minor, but it is a requirement about requirements. Once it is written down, the report refreshes itself on a schedule instead of depending on someone's heroics.

From wish to trusted dashboard in seven steps

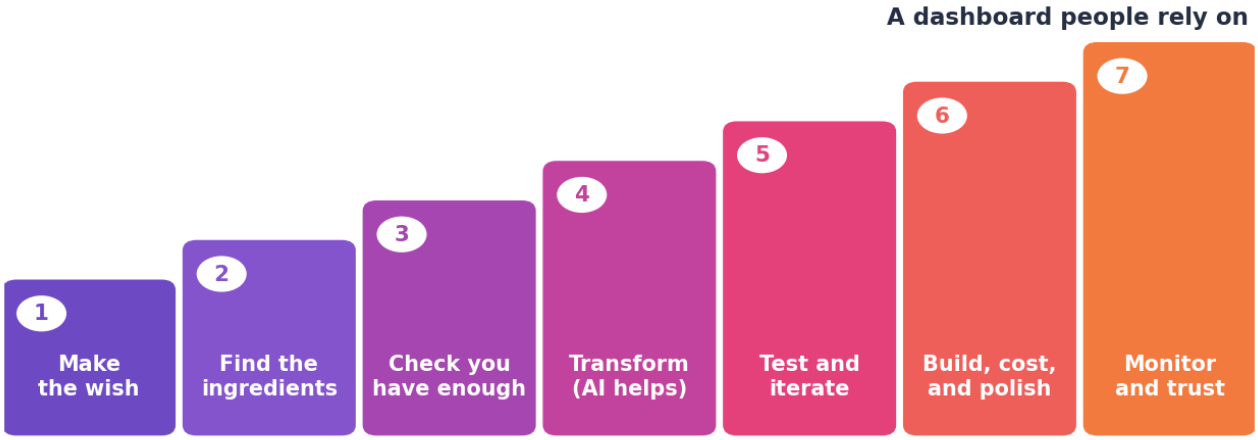


Figure 2. Seven steps from a wish to a dashboard people rely on.

Step	The question	You end with	Read
1. Make the wish	What do users want to see, and how fresh?	Output requirement	Section 4
2. Find the ingredients	Which data produces each output?	Input list	Sections 4, 9
3. Check you have enough	Can the data support the report?	Go or no-go	Section 4
4. Transform	How do we turn input into output?	Tested logic	Sections 10, 11
5. Test and iterate	Does it match reality and please users?	Sign-off	Sections 12, 13
6. Build, cost, polish	What will it cost, and is it easy to use?	A friendly dashboard	Sections 14 to 16
7. Monitor	Can users rely on it to decide?	Lasting trust	Section 17

Remember three things.
1. Output first. Decide what people want to see before touching any data.
2. No source, no report. If the data for an output does not exist, find out now, not after the build.
3. Trust is a feature. The dashboard must be easy to use and well monitored, so people rely on it to make real decisions.

Why it matters: follow this process and every team can focus on driving the business, instead of spending precious time on work that could have been automated.

If you are short on time, hand this document to your team. These two pages give them the whole picture. The chapters that follow cover the details they may not know yet, so they can dig into each one when they reach it.

1. Purpose and How to Use This Document

This playbook uses one concrete example, an ETL pipeline feeding an operations dashboard, to show what a process actually is. A process is not a pile of scripts. It is a written, repeatable way of getting from a business question to a trusted number on a screen, in which every step has an owner, an input, an output, and a check.

The core idea: the value you sell shifts from operating the work to designing, improving, and being accountable for the system that does it. The customer does not need you to run the pipeline every day. They need you because you know why it is shaped the way it is, and what to change when the business changes.

How to read it: Start Here gives the whole process on two pages. Sections 2 to 7 cover the fundamentals, the output-first method, a worked example, and roles. Sections 8 to 16 walk through each stage of the build, from defining the output to a dashboard people enjoy using. Sections 17 to 22 cover operating it, handing it over, and staying valuable afterward. The appendices are templates you can copy.

2. Why Process Beats Heroics

When only one person can run something, three problems follow.

Problem	What it looks like	Cost
Key-person risk	Only you know how the nightly load works	Buyers discount or refuse the business
Invisible quality	Numbers are right because you check them by hand	Errors appear the day you are away
No scaling	Every new customer needs your time	Revenue is capped by your hours

A process fixes this by moving knowledge out of your head and into documents, automation, and checks. The goal is not to remove yourself from the business. It is to remove yourself from the daily operation so you can work on the design, the customers, and the next thing.

3. Input, Transform, Output: The Fundamentals

Before any tools or technology, here is the simplest way to see what is going on. Every system that produces a useful result works the same way: something goes in, something happens to it, and something comes out. In data work these three parts have names. **Extract** is the input, **Transform** is what happens in the middle, and **Load** is the output. Together they are called ETL.

You already know this pattern from everyday life.

System	Input	Transform	Output
A kitchen	Ingredients	Cooking	A meal
A payroll run	Timesheets and pay rates	Calculate pay and tax	Payslips and payments
A spreadsheet	Raw numbers	Formulas	A report
An analytic dashboard	Ledgers, budgets, logs	Clean, join, calculate	Charts and tables

Once you see the pattern, a dashboard project stops being mysterious. It is a meal. Someone wants to eat it, the ingredients have to be found, and a recipe has to be followed. The same three questions work for any automation, whether it is a monthly report, a data migration, or a nightly integration between two systems.

What each part must answer

Part	The question it answers	What goes wrong
Extract (input)	Where does the data live, and how do we get it out?	No access, missing data, stale copies
Transform	What rules turn the input into the answer?	Hidden assumptions, silent errors
Load (output)	Where and in what form do people see the answer?	Pretty but unusable, or not trusted

Why the letters can mislead. ETL lists the parts in the order the data moves: input, then transform, then output. That is the right order for building. It is the wrong order for thinking. The people who need the dashboard care about the output first, and the output decides which input is needed. The next chapter turns the order around.

> **Two things to pin down with users:** the output they want to see, and the input that can produce it. Users know the first one best. The second is found by working backwards from the first, together with the people who own the data. Once both are clear, the transform in the middle becomes a well-bounded task instead of an open-ended one.

4. Think Backwards: Output First, Then Input

The cooking rule

A cook who shops first buys whatever looks good and then wonders what to make. Many data projects start the same way. Someone asks what data is available and builds charts from the answer. The result is a dashboard that is technically correct and practically ignored.

A better cook decides what to eat, then writes the shopping list. We do the same. First we make a wish about what we want to see. Then we reverse-engineer what it takes to grant the wish.

Step 1: Make the wish

Run a short wish session with the people who will use the report. Ask them to describe the report as if it already existed.

- What do you want to see? Describe it in your own words, and sketch it if you can.
- What decision will it help you make?
- Do you only want to look, or also play with it, for example by asking what happens if expenses fall by ten percent?
- Who will use it, and how often?
- How fresh must the data be: real time, hourly, daily, or monthly?
- How will you know the numbers are right?

The requirement about requirements: how fresh?

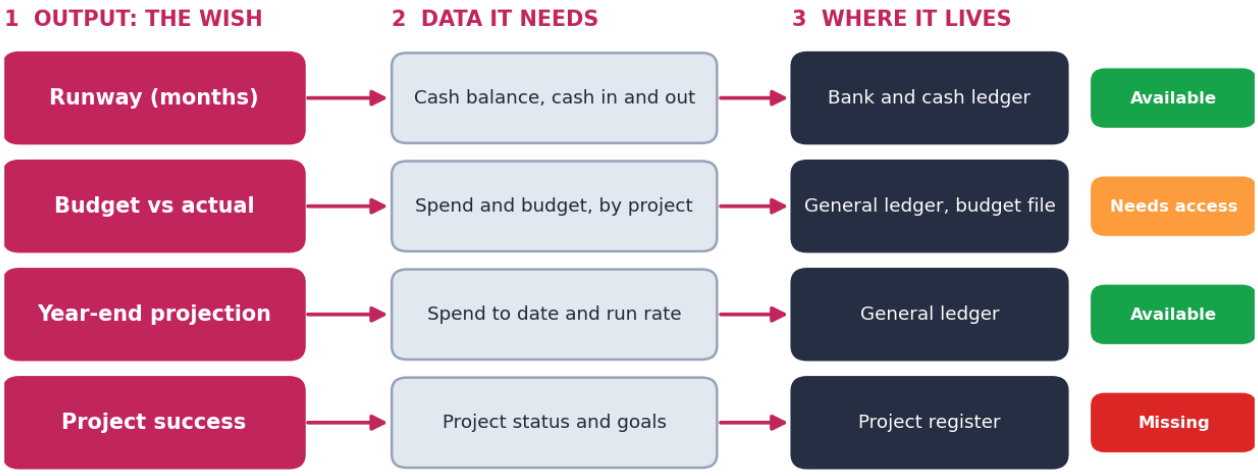
Every wish should say how often the data updates. This is a meta requirement, a requirement about the requirements, and it belongs on the first page of the output specification. Most people say real time when they mean not out of date, and the two cost very different amounts.

Frequency	Typically fits	Effect on the build
Monthly	Planning and review after the books close	Simplest and cheapest
Daily	Most management reporting	Low cost
Hourly	Operations monitoring	Higher cost
Real time	Alerts and live control	Highest cost and complexity

Stating the frequency up front is the spirit of this whole playbook: process instead of heroes. When the schedule is written down, the refresh can be automated and monitored. Nobody has to remember to update the report by hand, and nobody becomes the person who always does the monthly numbers.

Step 2: Reverse-engineer the input

With the wish written down, work backwards. For every number, label, and filter on the report, ask where it comes from.



Trace each output back to its data, then to the system that holds it. Then mark every source.

Figure 3. Working backwards from each output to the data it needs and the system that holds it. The example is illustrative.

1. List every number, label, and filter on the wished-for report.
2. For each one, name the business record it comes from, such as a ledger entry, a budget line, or a project record.
3. Name the system that holds that record and the person who owns it.
4. Note how the data will be reached and how often it changes.
5. Mark each source as available, needs access, or missing.

Step 3: Check you have enough

Status	Meaning	What to do
Available	The source exists and we can reach it	Proceed
Needs access	The source exists, but we cannot reach it yet	Request access now
Missing	No source exists for this data	Create it, change the output, or stop

> **No source, no report.** Do this check before building anything. It is the cheapest moment to discover a showstopper, and it turns a vague hope into a go or no-go decision.

When a gap appears, there are three honest options.

- Change the output so it needs less data.
- Get access to a source that already exists.
- Create the data, for example a simple controlled file with a named owner, such as a project register or a budget table.

The output requirement and the input list together are the contract between the users and the builders. Templates for both are in Appendix E and Appendix F.

5. Worked Example: The FP&A Runway Dashboard

This chapter follows one realistic request from start to finish. FP&A stands for financial planning and analysis: the team that builds budgets, forecasts, and performance reviews. The numbers in the charts and tables are illustrative, so you can see the shape of the work without any real company data.

Step 1: The wish

The finance lead describes what she or he wants to see, in plain words. Three wishes come out of the conversation.

View	The question it answers	How users interact
Runway	How many months of cash do we have?	View by month, filter by entity
What-if	What if we change budget or expenses?	Sliders with instant recalculation
Mid-year review	Are we on budget, and what should change?	Drill from total to project

In the mid-year review, the user looks back at the budget set earlier in the year and sees how much has been spent so far, a projection for the rest of the year, and how successful each project has been, so the budget can be adjusted.

The output requirement sheet

Item	Requirement
Audience	Finance lead, budget owners, executives
Decisions	Spending pace, hiring, budget changes at mid-year
Granularity	Monthly, by project and department
Refresh	Monthly, after the books close (no real time needed)
History	Last 12 months of actuals and this year's budget
Accuracy rule	Totals reconcile to the general ledger

Notice the refresh line. Monthly after the close is enough here. That one decision keeps the build simple and cheap, and because it is written down, nobody will ever need to refresh the report by hand.

Steps 2 and 3: Find the inputs and check coverage

What the output needs	Where it comes from	Status
Monthly cash in and out	Bank and cash ledger accounts	Available
Actual spend by project	General ledger, filtered by project	Available
Account groupings	Chart of accounts	Available
Budget by project and month	Budget workbook or planning tool	Needs access
Project status and goals	Project register	Missing
Forecast assumptions	Agreed assumptions table	Missing

Three of the six inputs are available today. The budget needs an access request, and two items do not exist yet: a project register and an assumptions table. This is the moment to find that out. The team decides to create both as simple controlled files with named owners before building anything. The answer is go, with two actions.

Step 4: Define the measures once

Measure	Definition
Net burn	Cash out minus cash in, per month
Runway (months)	Current cash divided by average monthly net burn (last 3 months)
Year-end projection	Spent so far plus expected spend for the remaining months
Variance	Budget minus actual

If net burn is zero or negative, runway does not apply, because cash is not running down. Each definition is agreed once and reused by every view, so the numbers never disagree between screens.

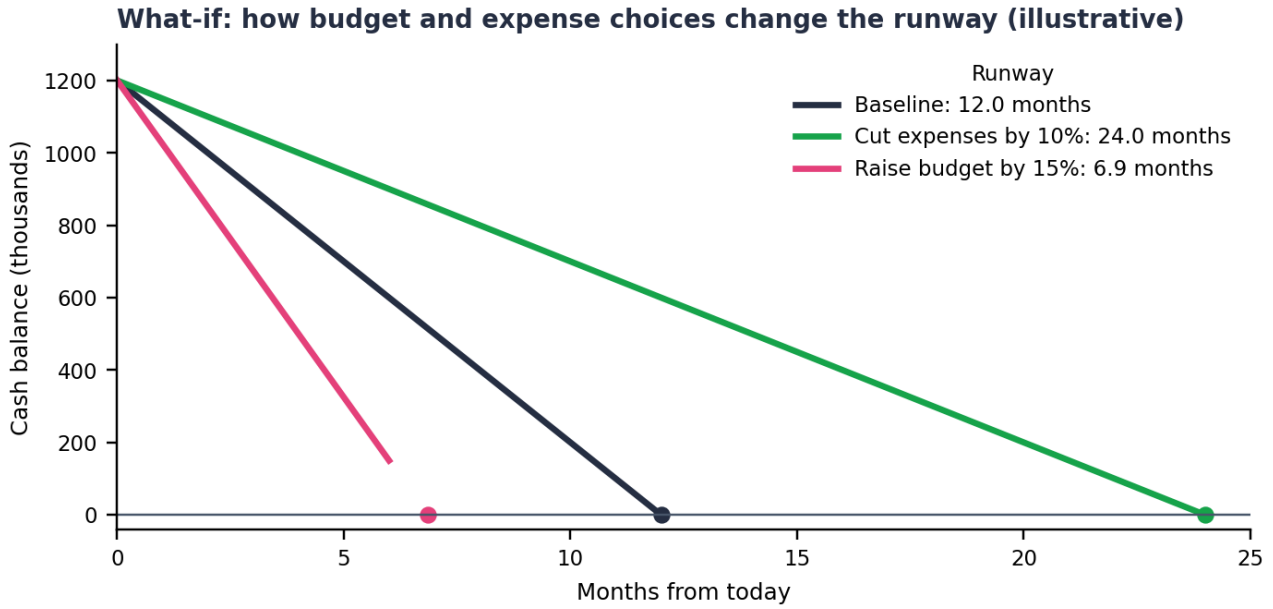


Figure 4. The what-if view. With cash of 1,200 and a monthly plan of 500 out and 400 in, the baseline runway is 12 months.

In this illustration the company holds 1,200 (in thousands) and burns 100 a month, so the runway is 12 months. Cutting expenses by 10 percent lowers the burn to 50 and doubles the runway. Raising the budget by 15 percent lifts the burn to 175 and cuts the runway to about 7 months. The user sees all three at once and can decide with

their eyes open.

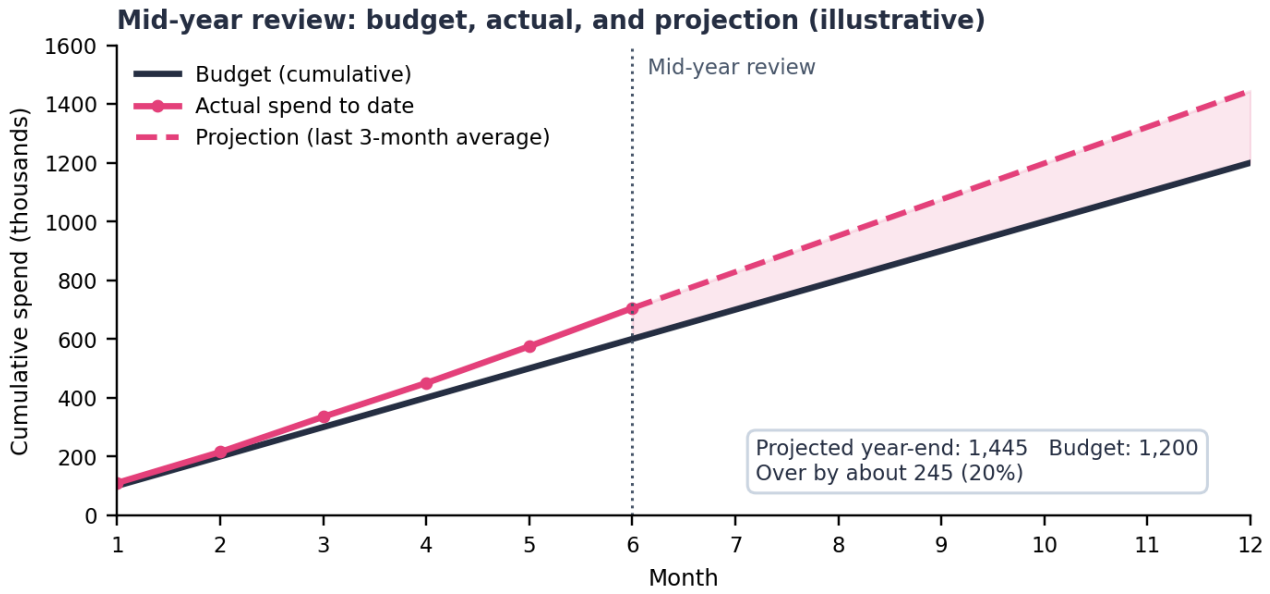


Figure 5. The mid-year review. Spending is running above the budget line, and the projection shows where the year will end.

At mid-year, 705 has been spent against a budget of 600 for the first six months. If spending continues at the recent three-month average of about 123 a month, the year ends near 1,445 against a budget of 1,200. The review then asks which projects explain the gap and whether budget should move between them.

Project	Budget used at mid-year	Signal
Project A	48%	On track
Project B	71%	Watch closely
Project C	95%	Over budget
Project D	30%	Under budget, check delivery

Budget used is only half the story. Success also needs a result measure, such as a milestone reached or a target met, and that comes from the project register created in Step 3.

Steps 5 to 7: Transform, test, polish, and monitor

- **Transform:** AI drafts the cleaning and joining logic from the output requirement and the source samples (Section 11).
- **Test and iterate:** totals are reconciled to the general ledger, then the finance lead tries the what-if sliders and asks for changes (Section 13).
- **Cost and polish:** a monthly refresh keeps the infrastructure small (Section 14). The first screen shows the runway in months, and the sliders respond instantly (Section 16).
- **Monitor:** the dashboard shows when the data was last refreshed and whether every check passed, so users can rely on it (Section 17).

6. The Pipeline at a Glance

Section 5 follows an FP&A dashboard from start to finish. The stage chapters also draw on a second example from operations, to show the same process in a different domain.

A data operations team wants a dashboard showing KPIs pulled from several enterprise systems, for example finance, CRM, collaboration tools, and equipment monitoring. Leadership wants one place to see the numbers and trust them.

The pipeline has five stages, followed by an operating loop.

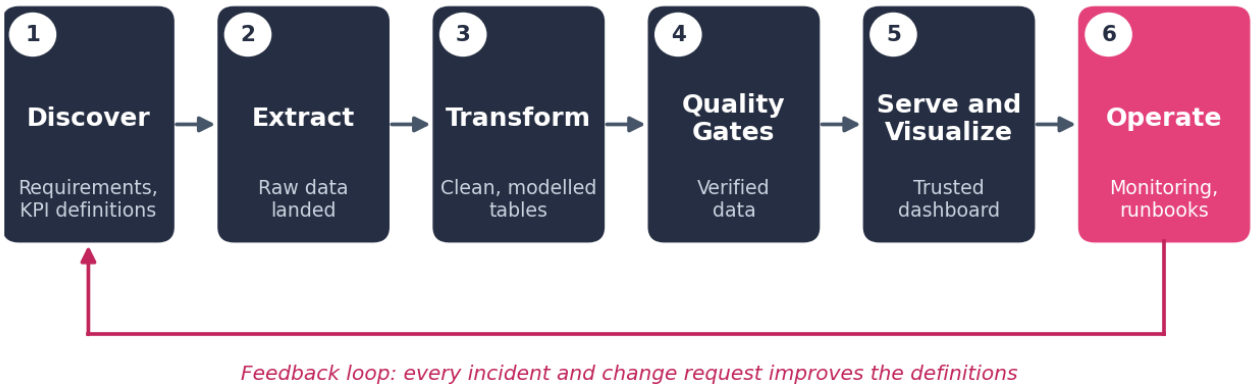


Figure 6. The pipeline: five build stages and an operating loop that feeds improvements back into the definitions.

Stage	Question it answers	Main output
Discover	What decisions will this support?	Requirements and KPI definitions
Extract	Where does the data live and how do we get it?	Raw data landed unchanged
Transform	How do we make it consistent and correct?	Clean, modelled tables
Quality Gates	How do we stop bad data reaching users?	Verified, trusted data
Load, Serve, Visualize	How do people see and act on it?	Curated data layer and dashboard
Operate	How do we keep it healthy?	Monitoring, runbooks, change control

7. Roles and Ownership

Every step needs a named owner, even on a small team where one person holds several roles. Roles are about accountability, not headcount.

Role	Responsibility
Business owner	Defines what the numbers mean and signs off on them
Delivery lead	Plans, tracks scope, manages risk and communication
Data engineer	Builds and maintains extraction and transformation
Analyst or BI developer	Builds the dashboard and semantic layer
Source system owner	Controls access to and changes in each source
Operator	Watches runs, handles alerts, follows the runbook

Write these names down at the start. Most failed data projects fail on unclear ownership, not on technology.

8. Stage One: Discover and Define

Goal: agree on what is being measured before anything is built.

This stage records the result of the output-first method in Section 4 as signed-off definitions.

Activities

1. Interview the business owner and two or three dashboard users.
2. List the decisions the dashboard should support. If a number does not support a decision, question whether it belongs.
3. Define each KPI in plain language, with formula, grain, source, owner, and refresh frequency.
4. Agree on what good looks like, such as thresholds and targets.
5. Record assumptions and open questions.

KPI definition template

Field	Example
Name	Order cycle time
Definition	Days from order received to service ready
Formula	ready_date minus received_date
Grain	Per order, rolled up by site and month
Source	Order management system
Owner	Operations manager
Refresh	Daily at 06:00
Target	30 days or less

> **Exit criteria:** every KPI has a signed-off definition, and the business owner has agreed the scope in writing.

Common mistakes: starting with chart types instead of decisions, and letting two teams use the same KPI name for different calculations.

9. Stage Two: Extract

Goal: get data out of source systems reliably, without changing it.

Principles

- Land raw data exactly as received. Never fix problems at this stage.
- Prefer incremental extraction using a modified date or change feed. Use full loads only for small tables or as a periodic reconciliation.
- Use service accounts with the minimum read permissions required.
- Respect source system limits, such as API rate limits and business-hour restrictions.

Source inventory template

Source	Method	Frequency	Volume	Owner	Risk
Finance system	API	Daily	Medium	Finance	Rate limits
CRM	API	Hourly	High	Sales ops	Schema changes
Collaboration tools	Graph API	Daily	Low	IT	Token expiry
Equipment monitoring	Message feed	Near real time	High	Network ops	Gaps in feed

What to record on every run: start time, end time, rows read, rows landed, watermark value, and success or failure. These become the raw material for monitoring later.

Exit criteria: each source lands on schedule, a failed run can be retried safely without duplicating data, and credentials are stored in a secrets manager, not in code.

10. Stage Three: Transform

Goal: turn raw data into consistent, correct, business-ready tables.

Section 11 shows how AI can help draft this work once the output and input are defined.

A useful pattern is three layers.

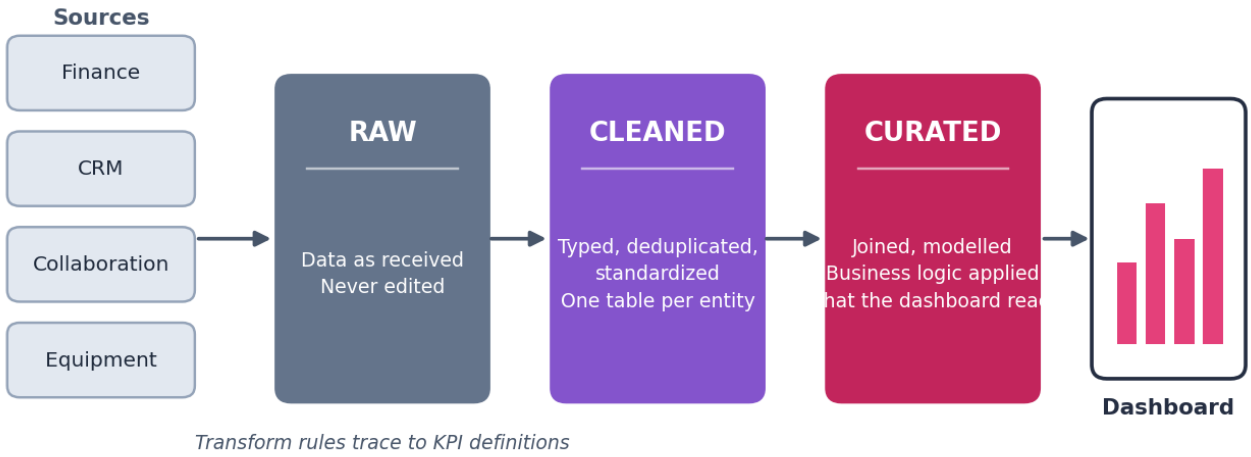


Figure 7. Three-layer data pattern: raw data is never edited, and business logic is applied only in the curated layer.

Layer	Contents	Rule
Raw	Data as received	Never edited
Cleaned	Typed, deduplicated, standardized	One table per source entity
Curated	Joined, modelled, business logic applied	What the dashboard reads

Core transformation tasks

1. Standardize types, time zones, currencies, and units.
2. Deduplicate using a clearly defined business key.
3. Handle missing values with an explicit rule, not silence.
4. Map codes to readable names using reference tables.
5. Apply business logic from the KPI definitions, and nowhere else.
6. Model for reporting, usually facts and dimensions.

Idempotency: running the same step twice must give the same result. This single property makes retries, backfills, and recovery safe, and it is one of the biggest differences between a hobby pipeline and a professional one.

Documentation rule: every business rule in the code must trace to a KPI definition. If you cannot point to the definition, either the code or the definition is wrong.

> **Exit criteria:** curated tables reconcile to source totals within an agreed tolerance, and all rules are documented.

11. Using AI to Build the Transform

Goal: use AI to draft the transformation quickly, while people and tests make sure it is right.

Once the output is defined and the input is confirmed, the transform stops being open-ended. You know what must come out and what goes in. That is exactly the brief a good developer needs, and exactly the brief AI works best from. This is why the transform is often the easier part of the project.

What to give the AI

- The output requirement: the views, the columns, the grain, and the definition of every measure.
- The input: table and column names, with a handful of sample rows from each source.
- The rules: KPI definitions, time zone, currency, rounding, and the layer rules from Section 10.
- A known answer: a few cases where you already know the correct result, for testing.

Example brief

```
GOAL    Produce the monthly cash and budget tables for the runway dashboard.
OUTPUT  Columns, grain, and the definition of each measure (net burn, runway).
INPUT   Table names, column names, and five sample rows of each source.
RULES   KPI definitions, time zone, currency, rounding, and the layer rules.
CHECKS  Totals must reconcile to the general ledger within the agreed tolerance.
RETURN  The code, a plain-language explanation, the assumptions made, and tests.
```

Who does what

AI can draft	A person must check
Cleaning and joining code	Join keys and filters match the business meaning
Mapping of codes and accounts to groups	The business owner approves the mapping
Tests and reconciliation queries	The tests cover the cases that matter
Plain-language documentation	The documentation matches the code

AI proposes, tests decide. Never accept transformation logic that is not backed by a test against a known answer, such as a reconciliation to the ledger totals. A confident explanation is not evidence. A passing test is.

Risks and guards

Risk	Guard
Confident but wrong joins	Reconciliation tests against source totals
Hidden assumptions	Ask for a list of assumptions, and review it
Sensitive data exposed	Use masked or sample data, and follow Section 19
Unreviewed changes	Version control and change control (Section 18)

12. Stage Four: Quality Gates

Goal: stop bad data before it reaches a dashboard.

A quality gate is an automated checkpoint between stages. If a check fails, the pipeline halts or flags the data instead of publishing it.

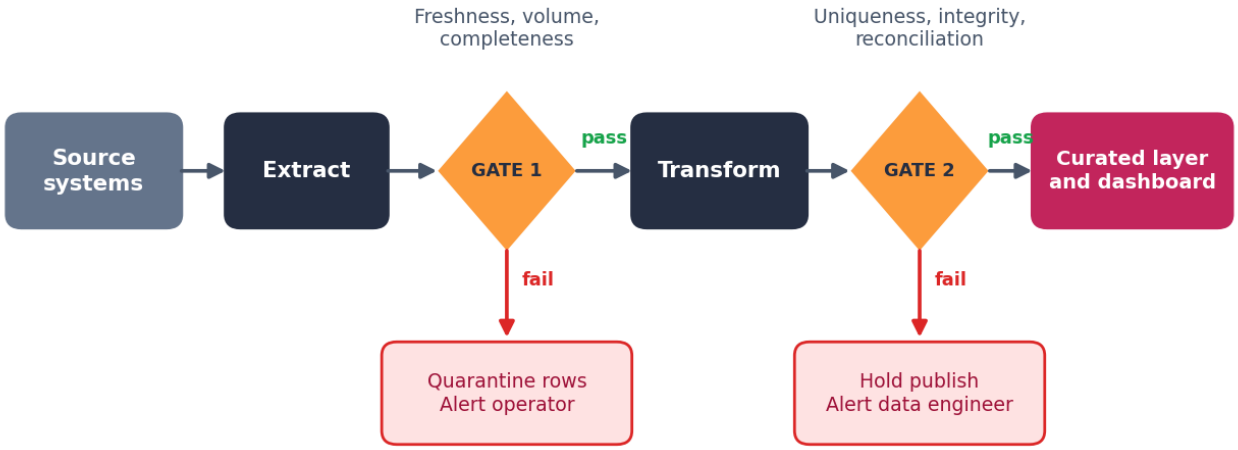


Figure 8. Quality gates sit between stages. A failed check quarantines rows or holds publishing instead of passing bad data on.

Types of checks

Check	Example	Action on failure
Freshness	Latest record is under 24 hours old	Alert, hold publish
Volume	Row count within 20% of the 7-day average	Alert, review
Completeness	Required fields are not null	Quarantine rows
Uniqueness	No duplicate business keys	Fail the run
Referential integrity	Every order links to a valid site	Quarantine rows
Reconciliation	Totals match source within tolerance	Hold publish
Range and sanity	No negative durations	Quarantine rows

Severity levels: decide up front which failures block publishing and which only warn. Too many blockers cause alert fatigue and people start ignoring them. Too few let bad numbers through.

A quality gate dashboard is a small companion view showing each check, its last result, and its trend. It builds trust because users can see the data is being verified, not just assumed to be correct.

> **Exit criteria:** every critical KPI has at least one automated check, and the failure path is tested deliberately, not just the happy path.

13. Test and Iterate

Goal: prove that the dashboard is right, and keep refining it until users say it answers their questions.

The transform is rarely perfect on the first attempt, and the wish is rarely complete after the first conversation. Treat both as normal. Plan for several short rounds of building, testing, and showing the result to real users.

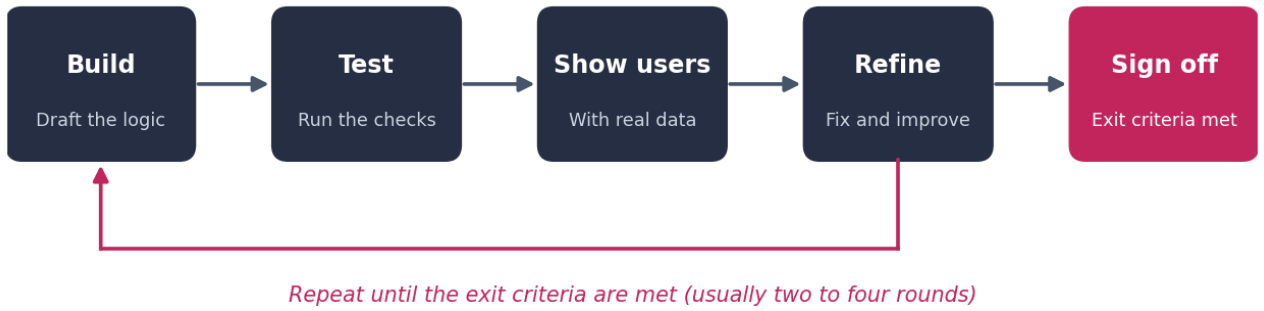


Figure 9. The iteration loop. Each round ends with real users looking at real data.

What to test

Test	What it proves	Who
Unit tests	Each rule works on small examples	Data engineer
Reconciliation	Totals match the source of truth	Engineer, business owner
Quality gates	Checks pass on real data	Automated
Performance	The dashboard is fast at real volume	BI developer
User acceptance	Real users can answer real questions	Business owner, users

How to run an iteration

- Show the dashboard with real data every week or two. Demos with sample data hide problems.
- Write feedback down as changes to the output requirement, so the wish stays the single source of truth.
- Keep a decision log of what changed and why.
- Park new wishes in a backlog. Finish the current round before opening the next.
- Stop when the exit criteria are met, not when the ideas run out.

> **Exit criteria:** reconciliation totals are signed off, quality gates pass on production data, and at least three real users have completed their key tasks without help.

14. Infrastructure and Cost Estimation

Goal: choose infrastructure that fits the requirement, and tell the business what it will cost before it is built.

The refresh frequency from Section 4 is the biggest lever. A report that updates monthly can run on small, simple infrastructure. A report that updates in near real time needs streaming, always-on compute, and more monitoring. The same dashboard can cost many times more depending on that one answer.

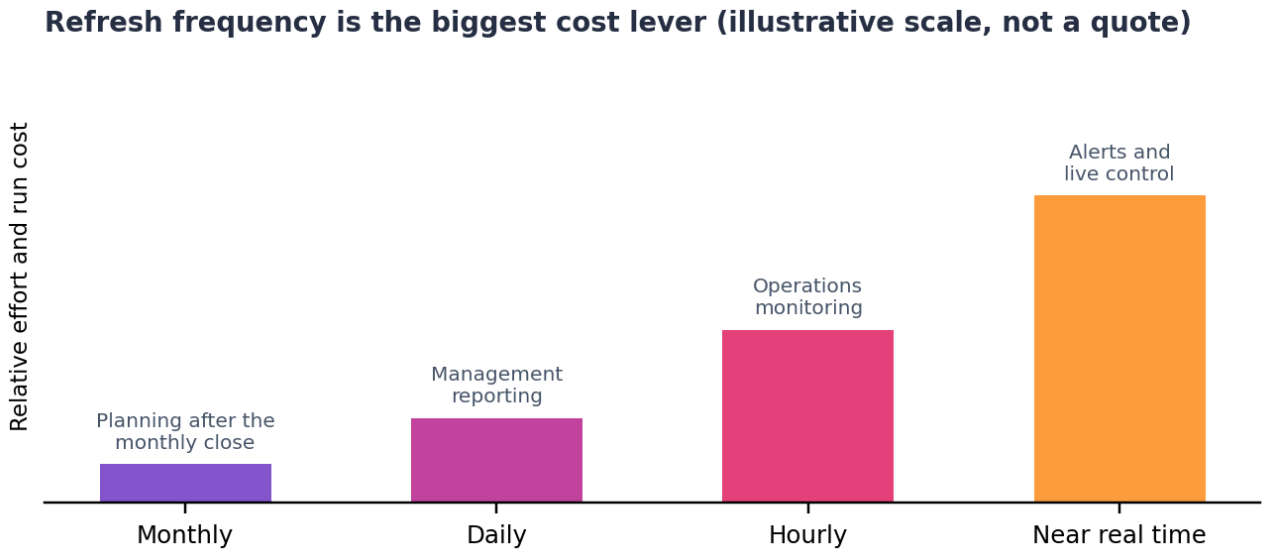


Figure 10. Higher refresh frequency means more infrastructure, more checks, and more support. The scale is illustrative.

Building blocks

Component	What it does	What drives the cost
Storage	Holds raw, cleaned, and curated data	Volume and retention
Compute	Runs the transformations	Frequency and complexity
Orchestration	Schedules and sequences the jobs	Number of jobs
Dashboard tool	Serves the report to users	Number and type of users
Monitoring	Runs checks and sends alerts	Number of checks
People time	Builds, supports, and improves it	Complexity and iterations

How to estimate

1. List the components the design needs.
2. Estimate data volume, growth, and refresh frequency from the output requirement.
3. Price each component using the provider's own pricing calculator.
4. Add people time for the build and for ongoing support.
5. Add a contingency margin that you and the business agree on.
6. Present the one-time build cost and the monthly run cost separately.

Cost summary template

Item	One-time	Monthly run
Storage		
Compute		
Orchestration and monitoring		
Dashboard tool and licences		
People time		
Contingency		

Offer options. Present the estimate at two or three refresh frequencies and let the business choose. Often the conversation changes from we need real time to monthly is fine, which saves money and removes risk.

15. Stage Five: Load, Serve, and Visualize

Goal: put trusted data in front of people in a form they can act on.

Making the dashboard easy and pleasant to use is covered in more depth in Section 16.

Serving

- Expose curated tables through a semantic layer so KPI logic is defined once and reused by every report.
- Control access by role. Not every user should see every site or customer.
- Keep dashboard queries fast, using aggregates or extracts where needed.

Dashboard design principles

1. **Lead with the answer.** Put the headline KPIs at the top, with status against target.
2. **One page, one purpose.** Split by audience if needed, such as an executive view and an operations view.
3. **Show context.** Every number needs a comparison: target, prior period, or trend.
4. **Make the data age visible.** Show a last refreshed timestamp prominently.
5. Allow drill-down from summary to detail, but design the default view for the 10-second glance.
6. **Use color sparingly and consistently.** Red should always mean the same thing.
7. Link every KPI to its definition so nobody has to guess.

Acceptance: the business owner reviews the dashboard against the original decisions list from Stage One and signs off. User acceptance testing uses real data and real users, not demo data.

16. Designing for the User

Goal: make the dashboard so easy to use that people choose it over their own spreadsheets.

User-friendly is not decoration. A confusing dashboard is a dashboard people do not trust, and a dashboard people do not trust is not used. This is one of the most important parts of the whole process, and it deserves real time in the plan.

Principles

1. **Answer first.** Open on the number people came for, such as the runway in months, not on a menu.
2. **Speak the user's language.** Use the words from the wish session, not system field names.
3. **One screen, one question.** If a screen answers two questions, split it.
4. **Make interaction safe.** What-if controls respond instantly, show the baseline next to the scenario, and always offer a Reset button.
5. **Guide the eye.** Use a clear hierarchy, consistent colors, and enough white space.
6. **Show the trust signals.** Display the data as-of date and the check status, and link every number to its definition.
7. **Keep it fast.** A page that takes ten seconds to load teaches people to stop opening it.
8. **Include everyone.** Do not rely on color alone, keep text readable, and test on the devices people actually use.
9. **Teach in one page.** Offer a short guide or tour for first-time users.

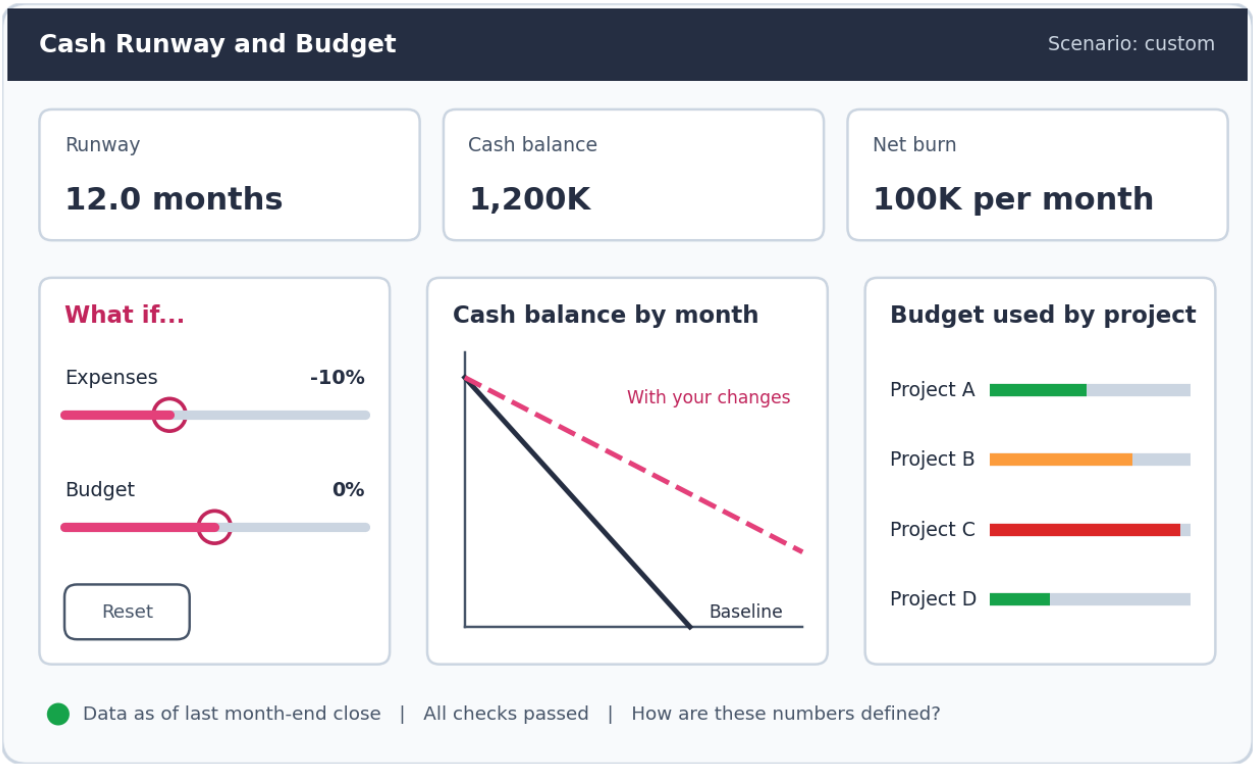


Figure 11. A wireframe of the FP&A dashboard: answer first, safe what-if controls, and trust signals in the footer.

Test it with real users

Ask five real users to complete three tasks without help, for example find the runway, run a what-if, and check where a number comes from. Watch quietly. Wherever they hesitate is the fix list.

Cosmetics matter, after function

Consistent fonts, spacing, and your brand colors make people believe the dashboard was built with care, and that belief supports trust. Polish the look once the function is right, not before.

> **The ten-second test:** a new user can find the headline number in ten seconds, run a what-if in under a minute, and say where the data came from.

17. Operating the System

A pipeline that is built but not operated is a liability. Operations needs its own design.

> **Why monitor?** so that users can rely on the dashboard to make accurate decisions. If people stop trusting the numbers, they go back to their own spreadsheets, and the whole investment is lost.

Scheduling and dependencies: use an orchestrator so stages run in order and downstream steps wait for upstream success.

Monitoring

Signal	Tool or approach	Who is alerted
Run failures	Orchestrator alerts	Operator
Quality gate failures	Gate dashboard and alerts	Data engineer
Late data	Freshness check	Operator, then source owner
Dashboard errors	Report refresh alerts	BI developer
Cost and performance	Usage reports	Delivery lead

Runbooks: for every alert there should be a short document saying what it means, how to check it, how to fix it, and when to escalate. A good runbook lets someone who has never seen the system resolve the issue. This is the main document that removes you from daily operations.

Incident handling: classify severity, communicate early with a short non-technical status message, fix, then write a brief review covering cause, impact, and prevention.

18. Change Control

Processes decay when changes are made casually. Define how change happens.

Types of change

Type	Example	Approval
Standard	Add a filter to a report	Delivery lead
Normal	New data source or KPI	Business owner plus delivery lead
Emergency	Fix for broken production run	Delivery lead, reviewed afterward

Every change should have: a description, a reason, a test plan, a rollback plan, and an updated document. Keep development, test, and production environments separate, and promote changes through them in order.

Versioning: keep code, configuration, and KPI definitions in version control. A KPI definition changing is a business event and should be announced, not slipped in quietly.

19. Security and Governance

- Least privilege access to every source and every output.
- Secrets in a vault, rotated on a schedule.
- Personal or sensitive data identified, minimized, and masked where possible.
- Audit trail of who changed what and who accessed what.
- Data retention rules, so you keep what you need and delete what you should.
- A data owner and steward named for each domain.

Good governance also protects you commercially. Customers in regulated industries ask for these answers before they sign.

20. Handover: Designing Yourself Out

This section answers the central question of the playbook: why customers still need you once the process runs without you. Handover is a deliverable, not an afterthought.

Handover package

1. Architecture overview, one page with a diagram.
2. Source inventory and KPI definitions.
3. Data dictionary for curated tables.
4. Runbooks for every alert.
5. Access and credential inventory, with owners.
6. Change control procedure.
7. Recorded walkthrough and a live training session.
8. A 30-day shadow period where the customer operates and you observe.

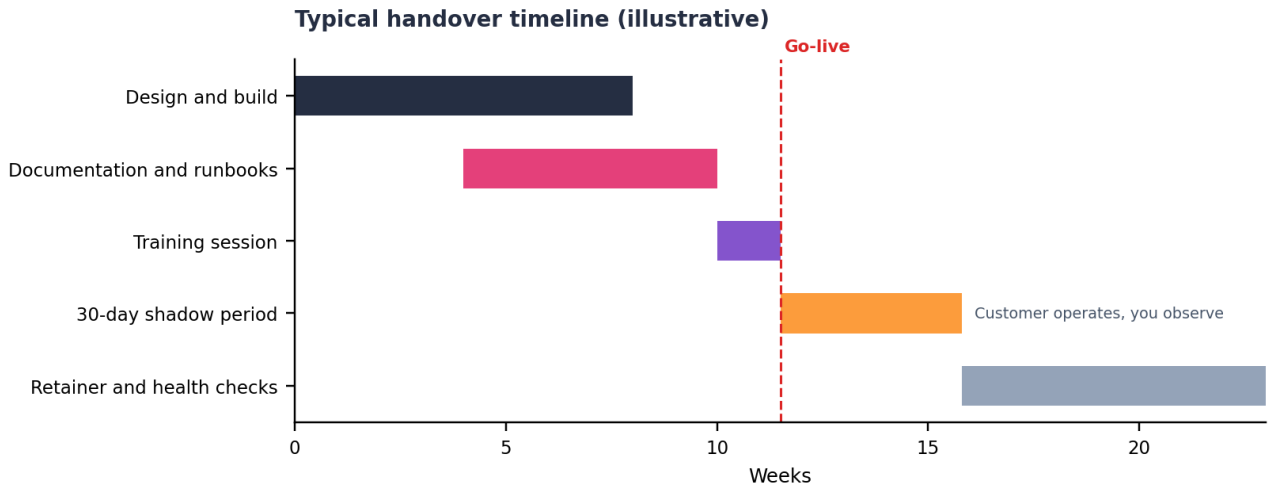


Figure 12. A typical handover runs from build to a 30-day shadow period, then moves to a retainer.

> **Handover test:** ask the customer's operator to resolve a simulated failure using only the documents. Where they get stuck is where the documents are incomplete.

What you still own after handover: the design authority, the roadmap, and the relationship. You are no longer the operator, but you are the person who knows why the system is shaped this way.

21. Staying Valuable When the Process Runs Without You

If the process does not need you, why do customers keep you? Because the value moves up a level.

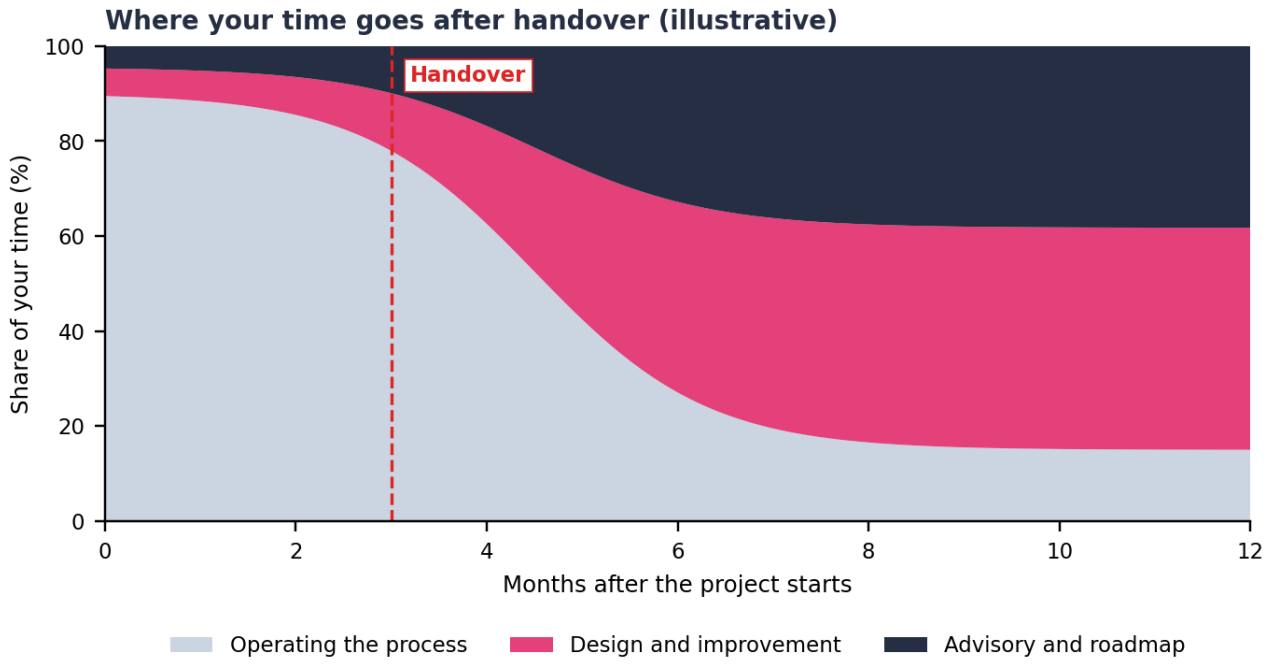


Figure 13. As operating work falls away, your time moves to design, improvement, and advice.

What you stop being paid for	What you start being paid for
Running the nightly load	Designing the system and its standards
Fixing routine failures	Improving reliability and reducing cost
Building one more report	Deciding which reports matter
Being on call	Being the trusted advisor on what to build next

Ways this turns into a business

- **Build fee:** a fixed price for designing and delivering the system and its handover package.
- **Retainer:** a monthly fee for roadmap, reviews, and priority support, not for operating.
- **Health checks:** a periodic audit of quality, cost, and security, with recommendations.
- **Change projects:** processes always change as the business does, and the person who designed it is the natural person to extend it.
- **Productized offering:** the playbook itself, packaged and repeated across customers.

The test of a good process is that the customer could run it without you, and still chooses to keep you because you make it better.

22. Metrics for the Process Itself

Measure the pipeline the way you would measure any service.

Metric	Meaning	Healthy sign
On-time delivery rate	Share of runs finishing before the SLA time	Above 99%
Data quality pass rate	Checks passed per run	Stable or improving
Mean time to detect	Failure to alert	Minutes, not hours
Mean time to recover	Alert to fix	Falling over time
Incidents per month	Production failures	Falling over time
Dashboard adoption	Active users per week	Growing
Operator independence	Incidents resolved without escalation	Rising

The last row is the one that tells you the process is truly transferable.

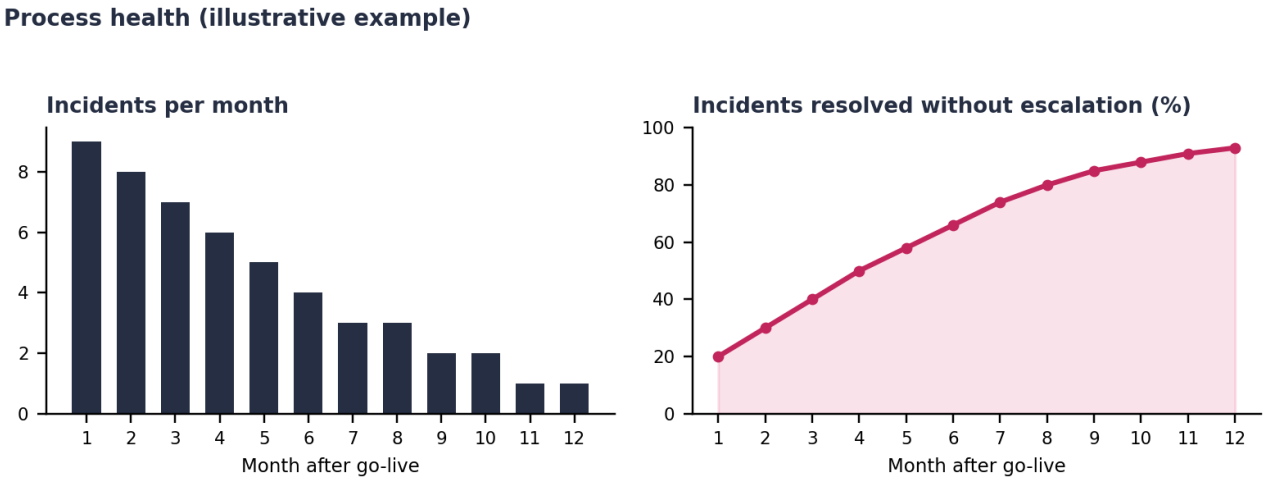


Figure 14. As the process matures, incidents fall and operators resolve more issues without escalation.

In Closing: Focus on the Business

A process turns a one-time effort into something that keeps working. Decide what people want to see. Find the data that makes it possible. Let AI help with the transformation. Test it with real users, make it a pleasure to use, and monitor it so people can rely on it to make accurate decisions.

Follow these steps and every team can focus on driving the business, instead of spending its precious time on work that could have been automated.

A good first week

1. Pick one report that people ask for every month.
2. Hold a short wish session and fill in the output requirement sheet (Appendix E).
3. Map each output to its data and mark every source available, needs access, or missing (Appendix F).
4. Decide go or no-go, and name an owner for every gap.
5. Ask AI to draft the transform, and write the tests that prove it right.

Appendix A: Project Kickoff Checklist

- [ ] Business owner and sponsor named
- [ ] Output requirement written and signed, including refresh frequency
- [ ] Input sources mapped and coverage checked (go or no-go)
- [ ] Decisions the dashboard supports listed
- [ ] KPI definitions drafted
- [ ] Source systems and owners identified
- [ ] Access requests submitted
- [ ] Environments agreed
- [ ] Success criteria and timeline agreed
- [ ] Communication plan and meeting rhythm set

Appendix B: Runbook Template

Field	Content
Alert name	
What it means	
Impact on users	
How to confirm	
Steps to fix	
How to verify the fix	
Escalate to	
Last reviewed	

Appendix C: Go-Live Checklist

- [ ] All quality gates passing on production data
- [ ] Reconciliation signed off by the business owner
- [ ] Access roles tested with real users
- [ ] Monitoring and alerts verified by triggering a test failure
- [ ] Runbooks complete and reviewed
- [ ] Rollback plan documented
- [ ] Handover training scheduled
- [ ] Support contacts published on the dashboard
- [ ] Trust signals visible on the dashboard (as-of date, check status)
- [ ] Usability test completed with real users

Appendix D: Glossary

ETL: Extract, Transform, Load. Moving data from sources, reshaping it, and storing it for use.

Grain: the level of detail one row represents, such as one order or one site per day.

Idempotent: safe to run repeatedly with the same result.

Semantic layer: a shared definition of metrics and relationships that every report reuses.

SLA: service level agreement, the promised timing or quality of a service.

Runbook: step-by-step instructions for handling a specific operational event.

Watermark: a stored marker, like the latest modified date, showing how far extraction has progressed.

Appendix E: Output Requirement Sheet

Fill this in with the users before any data work starts. Sign it off, and treat it as the single source of truth for the report.

Item	Answer
Report name	
Audience	
Decisions it supports	
Views (what users want to see)	
Interactions (view, filter, what-if)	
Granularity	
Refresh frequency	
History needed	
Accuracy rule	
Owner and sign-off date	

Appendix F: Source Mapping Worksheet

List every output field, trace it to its source, and mark its status as available, needs access, or missing.

Output field	Source and record	Owner	Status