THE PLAYBOOK

The Audit

How I proved that an automation everyone trusted was quietly missing about 3 in 4 of the client calls it was meant to handle, without reading a line of its code. The repeatable method behind that: count from the source, find the one rule behind the gaps, and cover by hand with an end date. Plus the honest chapter on when an audit is more ceremony than help.

Paul Prado Pacardo

Executive Assistant · Project Manager · Operations me.kabuhayan.app

Contents

Live for weeks, doing almost nothing the green dashboard, and the count that broke it

1 **Green is not done** why the status light is the wrong witness

2 **Count from the source** one row per item, one column per stage

3 **Find the rule behind the gaps** the method, with the decision rules that drive it

4 **Cover by hand, with an end date** wait, replace, or cover by hand

5 **How audits go wrong** failure modes, and how to catch each one

6 **When not to audit** the cases where counting is the wrong move

Count it before you trust it closing

The audit checklist and the live artifact from this story

OPENING

Live for weeks, doing almost nothing

A small team relied on an automation that turned client calls into tasks. Someone recorded the call, the automation filed it, read it, and created the follow-up tasks for the people who owned the account. It had been "live" for weeks. Nobody had a reason to doubt it.

> THE COMPLAINT, AS IT ARRIVED
>
> "This call didn't get tasks." Said once, about one call, then fixed by hand and forgotten.

Every operator who inherits an automation built by someone else meets the same quiet fork. It rarely looks like a decision. It looks like an ordinary Tuesday with a dashboard that says everything ran.

THE FORK IN THE ROAD

THE SAFE MOVE

Trust the green dashboard. Runs completed, no errors, nothing flagged. Treat each complaint as a one-off, fix it by hand, move on. Nobody will blame you for believing the status light.

THE MOVE I MADE

Count outputs against inputs. List every call from the source, not from the automation, and mark what actually came out the other end. Slower, a little awkward, and impossible to argue with.

Figure: the decision that separates an operator who reports problems from one who proves them.

The count took an afternoon. Roughly 3 of every 4 client calls had produced nothing. Whole types of call produced zero. And the reason was one rule: the automation only acted where a project board already existed, so anything new or informal fell through without a single error.

This playbook is that afternoon, turned into a method you can run on any automation you rely on but did not build. It ends with the honest chapter: when an audit is the wrong move, because a method that claims to always apply is a slogan.

CHAPTER 1

Green is not done

An automation's dashboard answers one question: did it run? It almost never answers the one you care about: did the work happen? Those are different questions, and the gap between them is where silent failures live.

A run report describes the automation. It counts what the automation saw and what it did with it. If an input never reached it, or reached it and was skipped by a rule, the run still completes. Green.

A downstream count describes the work. It starts from what should have happened and checks whether it did. It cannot be fooled by the automation, because it does not ask the automation.

> THE BET
>
> An automation that reports success is not evidence of success. Only the record at the far end is.

I learned this twice. The second time was in a back office, where a new timekeeping app told staff their clock-outs had "synced". Meanwhile the back office watched a growing pile of shifts with no end time. Each one was treated as one person's mistake and chased with one email.

What changed it was boring: counting the open shifts every day, by date, and noting the names that repeated. Within a week the scattered complaints became a named pattern that could be reported upward with evidence attached. The app's own status had said "synced" the whole time.

The status light is a witness with an interest in the outcome. You would not let a vendor audit their own invoice; do not let an automation audit its own output.

CHAPTER 2

Count from the source

You do not need the code, the logins behind it, or the person who built it. You need a list of what went in and a way to see what came out. The tracker that does this is deliberately plain.

ROWS

Every input

One row per item that should have flowed through, exported from the source system: the calendar, the form, the inbox.

COLUMNS

Every stage

One column per step it should pass, ending at the output a human actually uses, not an internal file.

TAGS

Every type

Client, prospect, internal, new account. Gaps hide inside averages; types expose them.

Each cell gets yes, no, or n/a, with a link to the evidence. A short invented sample of what the first page looks like:

ITEM	TYPE	RECORDED	FILED	TASKS	CREATED BY
Weekly sync, Client A	Client	Yes	Yes	Yes	Auto
Kickoff, Client B	Client	Yes	Yes	No	
Review, Client E (new)	Client	Yes	No	No	
Intro call, Prospect D	Prospect	Yes	Yes	No	
Team standup	Internal	Yes	Yes	None needed	

The source rule is the whole method. If you build the list from the automation's own log, you only see what it already saw. The items it never noticed are exactly the ones you are looking for, and they will not be there.

> THE RULE
>
> Count from the source, never from the automation's own log. Build the tracker before you form an opinion.

CHAPTER 3

Find the rule behind the gaps

Complaints arrive one at a time, so they get fixed one at a time, and the pattern stays hidden. The audit's job is to turn a pile of anecdotes into one sentence the owner of the automation can act on.

Step 01

Fix a window

One to two weeks. Export every input from the source for that window, not the ones you remember.

Step 02

Mark every cell

Yes, no, or n/a per stage, with evidence. "None needed" is a real answer when an input truly needs no output.

Step 03

Compute the gap rate per type

Not just overall. An overall rate of 1 in 4 can hide a type at zero and another at 3 in 4.

Step 04

Write the one rule that explains it

"Only acts when a project board already exists." Then test it on two gaps and one success. If it fails any of the three, rewrite it.

Step 05

Send the rule, not the complaints

A short note: the rule, the evidence rows, the fix you are asking for. Then re-run the audit on a fresh window once the fix ships.

> **Above about 1 in 10, not trusted.** Any type with a gap rate above roughly 1 in 10 is covered by hand until a fix is proven.

> **Below it, watched.** Keep the daily count; act only if it rises.

> SEE IT YOURSELF
>
> A free version of this tracker is live at **me.kabuhayan.app/tool-pipeline-gap-auditor**. Paste items with a type and yes or no per stage; it returns the gap rate per type, the likely break-point stage, and a draft rule statement.

CHAPTER 4

Cover by hand, with an end date

Once the rule is known, the fix belongs to the owner of the automation, and their time is not yours to schedule. Meanwhile, work is being lost every day. There are three honest options.

OPTION	WHAT IT COSTS	WHAT IT RISKS
Wait for the fix	Nothing extra	People lose tasks every day until it ships
Replace the automation	A rebuild	Duplicated work and a turf fight you do not need
Cover by hand, send precise fixes	Some time each day	The cover quietly becoming permanent

I took the third. A short daily check at the end of the shift, one batch, covering only the types that failed the threshold. It protected the team at once, and it turned the conversation with the owner of the automation from complaints into specific rules.

CHECK BY TAG FIRST

Before creating anything by hand, look for what the automation already made, by its tag or marker, not by keyword search. Keyword search misses renamed items and you create duplicates.

DUPLICATES KEEP THE AUTOMATION'S ITEM

If both exist, keep the automatic one, move your detail into it as a comment, delete yours. The automation's item is the one its fix will maintain.

> STATED PLAINLY
>
> Manual cover is temporary. It carries an end condition: the fix confirmed on a fresh window. No end condition, no cover.

Worth being precise about what this is evidence for. One automation, one team, one rule that happened to explain most of the gaps. Not every audit finds a single clean rule. What generalises is the shape: a count from the source beats any number of anecdotes, and a rule statement gets a fix where a complaint gets an apology.

CHAPTER 5

How audits go wrong

An audit is a small system of its own, and it fails in predictable ways. Each of these happened to me or nearly did. Each has a cheap catch.

> **Counting only what you remember.** The calls you recall are the ones that went well enough to be memorable. Catch it by exporting the full input list from the source, every time.

> **Calling "no output" a failure when none was needed.** A standup with no follow-ups is not a miss. Catch it with an explicit "none needed" mark, so it is a decision, not a blank.

> **Fixing gaps one at a time.** It feels productive and hides the pattern. Catch it by requiring a written rule statement before any fix request goes out.

> **Manual cover becoming permanent.** Nobody decides this; it just happens. Catch it with a dated re-audit on the calendar the day the cover starts.

One more, from the back office, because it is the subtlest. A hold queue of hundreds of rows sat under a single error label. The instinct was to report it as one problem with one owner. Reading the message text row by row showed several distinct errors, each owned by a different person, some fixable only by the supervisor.

> THE TRAP TO AVOID
>
> Do not classify by the queue label. A label tells you what the system thinks is wrong, not what is wrong. Classify by the real message first, then count, then route each class to the one person who can fix it.

The same applies to your own tracker. If "none needed" starts appearing on rows that used to be misses, the audit is drifting toward what you hope is true.

CHAPTER 6

When not to audit

A method that claims to always apply is a slogan. There are real cases where a full audit is ceremony, or worse.

> **When the source cannot be listed.** If there is no independent record of what went in, you cannot count against it. Fix the input record first; an audit built on the automation's own log is theatre.

> **When a miss costs less than the count.** An automation that tidies a shared folder can miss a few files a week and nobody suffers. Spot-check it monthly and spend the time elsewhere.

> **When the owner already knows.** If the owner of the automation has named the rule and a fix date, a fresh audit adds nothing. Cover the gap and re-audit after the fix.

> **When the failure is loud.** Errors, alerts and crashes need a fix, not a tracker. This method is for the quiet kind, where everything says it worked.

> **When you are not the person who should hold it.** If the work belongs to another team, send them the count and the rule. Do not quietly take over their cover.

Even in these cases, half the method survives: **pick one downstream count and look at it daily.** Records missing a field, items stuck in a status. A rising count is your alarm, whatever the status light says.

> THE TEST TO APPLY
>
> Would a missed output from this automation cost someone real work, and can you list its inputs without asking it? If both, audit. If not, set a daily count and stop there.

CLOSING

Count it before you trust it

Before the audit, one quiet decision: believe the dashboard, or count. Everything in this playbook is downstream of choosing to count, from the source, by type, with a rule at the end instead of a complaint.

The fix itself was not mine to make. What I could make was a sentence precise enough that the owner of the automation could act on it, and a daily cover that meant nobody lost work while they did. That is most of the job.

The tracker from this story did not stay a one-off. It became a free tool on this site, which is its own lesson: the check you build for one automation is the check you will want for the next one.

Paul Prado Pacardo

Senior Executive Assistant, Operations and Project Manager. Ten years supporting C-level leadership. Available for remote work.

me.kabuhayan.app · linkedin.com/in/paulpacardo · pacardopaul18@gmail.com

THE TOOL

The audit checklist

Run this on any automation you rely on but did not build. If a box will not tick, chapter 6 applies.

BEFORE YOU COUNT

- [ ] Window fixed (one to two weeks), full input list exported from the source
- [ ] One column per stage, ending at the output a human actually uses
- [ ] Every input tagged by type: client, prospect, internal, new account

WHILE YOU AUDIT

- [ ] Every cell marked yes, no or n/a with evidence; "none needed" recorded on purpose
- [ ] Gap rate computed per type; anything above about 1 in 10 marked not trusted
- [ ] One rule per gap type, tested on two gaps and one success
- [ ] Fix request sent to the owner of the automation: rule, evidence rows, the fix asked for

WHILE IT IS BEING FIXED

- [ ] Daily cover running, one batch at end of shift, failing types only
- [ ] Existing items checked by tag before anything is created by hand
- [ ] Duplicates resolved: keep the automation's item, move detail in, delete yours
- [ ] Re-audit dated on a fresh window; cover ends when the fix is confirmed

THE LIVE ARTIFACT

The tracker from this story is free at **me.kabuhayan.app/tool-pipeline-gap-auditor**, alongside the other operations tools at me.kabuhayan.app/tools. Paste your items, mark the stages, and it names the break point per type.