Call Center Workflow Diagnosis

1. Problem Framing

The actual problem

The call handling process was built when manual was the only option. Today, agents summarize calls into notes, force them into yesterday's 40-code dropdown, and email follow-ups to internal teams. Three manual steps that compound into a slow, error-prone workflow. The cost lands on customers (longer hold times, wrong team), the product team (losing signal on bugs), and the agents themselves (fighting the system instead of solving the call).

What I'd need to know to confirm

- How has the disposition code distribution shifted over 24 months?
- How often do agents fall back to "Other" or other catch-all codes?
- How often do follow-ups land on the wrong team and need re-routing?
- Is the call-center data actually useful to the product team, or too noisy to act on?

What I'd request from the company

Data:

- Monthly disposition code distributions, last 24 months
- "Other" or catch-all code usage by month and by agent
- Records of where follow-up items from calls were sent first, and how often they had to be re-routed
- Product team's intake log from the call center, with any feedback on usefulness
- Product release calendar, last 24 months (to correlate call spikes with releases)
- Product downtime and incident log (to separate system-driven volume from categorization issues)
- A sample of recent call recordings: a mix of well-handled and flagged calls. The contrast reveals where the system breaks.

Conversations:

- Brief sessions with the product team and QA team. Their qualitative feedback captures what the data won't surface: whether the signal coming out of the call center is actually useful to them.

2. Synthetic Data Analysis

Dataset overview

Across 300 call records over 24 months, 22 of 42 codes saw meaningful use (5+ calls). 5 used 1-4 times. 15 never used at all in 24 months. More than a third of the dropdown is dead weight.

Codes that cluster together

- **Access cluster (40 calls).** Login Issue (14), Password Reset (12), Account Access (14). Three codes for one problem: a customer can't get in.

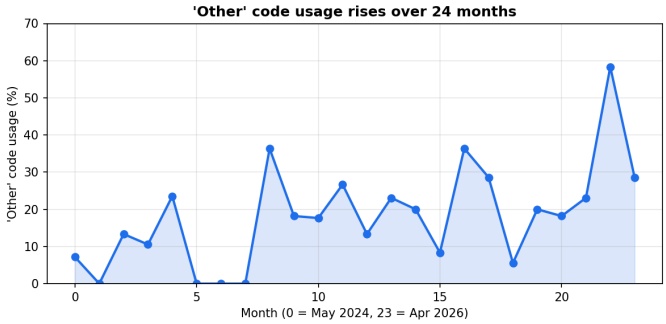
- **Bug-flavored cluster (92 calls across 8 codes).** Mobile App Crash, Mobile App Bug, Website Error, System Error, Notification Not Received, Failed Transaction, Mobile Deposit Failed, sometimes Login Issue. Software defects, coded by where they surfaced. The product team can't aggregate the signal because it's spread across 8 codes.
- **Catch-all cluster (67 calls, 22% of total).** "Other" (55), "General Inquiry" (7), "Account Question" (5). The agent didn't know what to pick.
- **Stale cluster (15 codes never used in 24 months, plus 5 used fewer than 5 times).**

Where the longest customer wait times are

Customer wait time = business clock time from end of call to issue closed. Top codes by average wait: ACH Transfer Question (1.7 days), General Inquiry (1.5), Login Issue (1.1), Bill Pay Issue (1.1), Mobile App Bug (1.0). But the slow codes aren't really the problem. The bigger driver is whether the call gets routed correctly the first time. Re-routed calls average 2.0 days; correctly-routed calls average 0.5. Where the call ends up matters more than what the call is about.

What the data suggests about the real friction

- **The categorization system is drifting fast.** "Other" usage rose from 7.5% in Q1 to 29.1% in Q4. The dropdown is no longer describing reality.
- **The drift isn't uniform.** Three agents drive disproportionate "Other" usage (44%, 42%, 39%) against a team average of 17%. Either less training or call types the codes can't capture.
- **Routing failures are where the real cost lives.** 19% of calls land on the wrong team and need re-routing.
- **Bug-flavored spikes track product releases.** Three bug codes spiked during product release windows: Mobile App Crash, Notification Not Received, Failed Transaction. Each returned to baseline within 2 months. These spikes were only visible in retrospect. Nothing flagged them in real time.



3. Recommendation

The bet

Build an AI-generated post-call summary that replaces the disposition dropdown, the manual case note, and the routing email. One AI pass at the end of every call produces three outputs:

- A structured summary (the case note)
- A suggested disposition code, with confidence
- A recommended routing team, based on call content

The agent reviews and adjusts before submitting. Human in the loop.

Why this beats fixing each piece separately

Three problems could mean three tools. One AI pass at the end of every call addresses all three: the summary becomes the case note, feeds the code suggestion, drives the routing recommendation, and surfaces the bug signal for product.

What success looks like

- **Agent post-call documentation time drops to ~30 seconds of review per call.** Each call gets a draft case note instantly.
- **Re-routing rate drops from 19% to under 5%.** AI uses call content to decide routing, not the agent's pick from a stale dropdown.
- **"Other" code usage drops from 29% to under 10%.** Better still: the structured summary makes the dropdown optional.
- **Customer wait time for re-routed cases drops from 2.0 days toward the 0.5-day baseline.** Each re-route avoided saves ~1.5 business days.
- **Product team signal quality improves.** Bug-flags and structured fields replace noisy code-based aggregation.

4. Tool Transparency

Tools used. Claude Code as primary thinking partner and draft writer. Python (csv, random, matplotlib, openpyxl, python-docx) for data generation, analysis, and spreadsheet output. Google Drive API for document production.

How I used them.

- Claude for problem framing dialogue and first drafts of every section.
- Python (scripted with Claude's help) to generate a 300-row synthetic dataset with patterns baked in: drift, agent variation, re-routing rate, code clusters, release-driven spikes, and zombie codes that exist in the dropdown but never get selected.
- Python plus matplotlib to analyze the dataset and produce the chart.

Where I applied my own judgment.

- The framing came from my own diagnosis. The product team angle, the customer hold-time cost, and the release-spike pattern as a data point were additions I made.
- I specified which hypotheses the synthetic data should test, and at what scale.
- I caught that the dataset didn't reflect a realistic dropdown. Real dropdowns always carry codes nobody picks. I asked for zombie codes (in the system but never selected) to be added so the analysis would surface that pattern.
- I rewrote significant portions of Claude's drafts to make every line earn its place. Edits enforced plain English, removed consultant vocabulary, and reframed confusing wording.
- I cut the brief from five pages to under three by identifying redundancy across sections.
- All structural decisions (what to keep, what to cut, what order) were mine.

What I did not independently re-execute. The math behind the analysis. I checked patterns at a high level and reframed metrics where the units were misleading. For example, Claude initially expressed resolution time in hours per call, which read as agent labor time. I converted the metric to customer wait time in business days. I did not re-run the underlying statistics by hand.