10

WHITE PAPER · WORK, ADOPTION & JUDGMENT

The Exception Path Is the Real Automated Workflow

Design recovery before declaring a workflow automated

Marco Policani

Enterprise Portfolio · PMO · AI Operating Governance

policani.net · linkedin.com/in/marcpolicani

July 2026

EXECUTIVE SUMMARY

A workflow is governed when its exception path has an owner, a boundary, an evidence trail, and a recovery decision.

Automation removes the routine cases, so what remains for humans is by construction the ambiguous, malformed, conflicting, novel, and unsafe — a small share of volume carrying most of the risk. This paper designs that stream as the real workflow: classified by cause, contained before consequence spreads, routed with context, owned twice, recovered fully, and mined quarterly for causes worth retiring.

01 Replace the manual-review bucket with a cause taxonomy; frequency shows the envelope's edge and recurrence shows what upstream fix would pay.

02 Design against the plausible guess — a workflow that never raises exceptions is either lucky or quietly guessing.

03 Measure causes retired, not volume suppressed; a class that vanished because the threshold moved is risk transferred downstream.

The argument

The happy path demonstrates throughput. The exception path demonstrates whether the operation can detect what falls outside the rules, route it to an accountable owner, bound the consequence, preserve evidence, recover, and learn.

Whenever a team tells me their workflow is automated, I ask to see the exception queue. The reaction to the question usually answers it. On one engagement, the proudly automated invoice-intake process turned out to have a companion inbox — literally a shared mailbox — where everything the automation could not handle had been accumulating for five months. Twelve hundred items. No classification, no owners, no service expectation, no record of what had been tried. Three people worked the mailbox part-time, in the order things arrived, and the oldest items included two regulatory notices and a customer dispute that had since become a formal complaint. The automated workflow processed ninety-one percent of its volume beautifully. The operation's actual risk, cost, and reputation lived entirely in the other nine.

That distribution is the rule, not the anecdote, and it follows from what automation is. Moving the routine cases out of human hands means the cases remaining in human hands are — by construction, not by accident — the ambiguous, the malformed, the conflicting, the novel, and the unsafe. The exception stream is where the hardest judgment concentrates, where the compliance exposure hides, and where customer trust is actually won or lost. Yet automation proposals describe it in a sentence ('exceptions route to manual review'), staff it as an afterthought, and measure it not at all.

This paper inverts the design order: the exception path is the real workflow, and the happy path is the optimization. It sets out a six-part path — classify, detect and contain, route with context, own the response, recover to a known state, learn and redesign — as the standard a workflow must meet before 'automated' is an honest description.

NIST's risk-management framework supplies the govern-map-measure-manage frame, and Microsoft's red-team experience shows that failure-finding must be deliberate and resourced [1][2]; the six parts are what the mailbox engagement — and the queues that came after it — taught me to require. The paper is written for workflow owners, automation leaders, AI program owners, operations executives, and risk teams.

The economics of the other nine percent

Why does the exception path decide the economics? Because its costs are per-case expensive and its failures are per-case consequential. A routine case costs seconds of compute; an exception costs specialist minutes or hours, plus the routing tour it takes to find the specialist, plus the context reconstruction once it arrives. A routine failure is a retry; an exception mishandled is a lapsed obligation, a wrong payment, an angry customer, or a regulator's letter. Small percentages of volume, large percentages of consequence — the asymmetry is structural.

The asymmetry compounds as automation succeeds. Raise the happy path from ninety to ninety-six percent and the queue shrinks — but the cases still in it are now the hardest six percent, the ones the improved model still could not touch. Every gain in coverage distills the residue. An operation that staffs and skills its exception handling for last year's mix will find this year's mix harder per case, which is why 'the queue is smaller now' and 'the queue is fine now' are different claims, and only the second one needs evidence.

There is also a visibility trap built into the accounting. Happy-path metrics are automatic — counts, latencies, rates, all free from the pipeline. Exception metrics require someone to decide the queue is worth instrumenting. So the workflow's dashboard fills with its best news while its risk concentrates where the instruments are not, and leadership reads accurate numbers that describe the safe fraction of the operation. The mailbox with twelve hundred items had never appeared on any dashboard. Nobody had decided to hide it. Nobody had decided to look.

The six-part exception path

The six parts run in operating order — what the case is, how it is caught and held safe, how it reaches the right person, who answers for it, how the world gets restored, and how the system gets better. Each part has a characteristic failure and a working design. Together they are the difference between an exception path and an exception pile.

EXHIBIT 1

Six parts turn an exception pile into an exception path

1. **Classify exceptions**
   An exception taxonomy should describe why ordinary processing is unsafe or impossible and what kind of response is required.
2. **Detect and contain**
   The system must recognize boundary conditions early and move into a safe state.
3. **Route with context**
   A human handoff needs the case, reason, evidence, attempted actions, affected systems, and deadline.

4 **Assign response ownership**
Each class needs an owner for the case and an owner for the recurring condition behind it.

5 **Recover to a known state**
Resolution must restore data, systems, customer commitments, and workflow state—not simply close a ticket.

6 **Learn and redesign**
Exception evidence should change rules, data, interfaces, training, permissions, and the definition of eligible work.

Classify exceptions

The first part replaces the generic manual-review bucket with a taxonomy of why ordinary processing was unsafe or impossible: missing or conflicting data, low confidence, policy conflict, dependency failure, unauthorized or unusual request, suspected harm, and the genuinely novel case. Each class implies a different response urgency, a different skill, and a different fix at the source — which is exactly the information the single bucket destroys.

Classification is what converts the queue from a pile into a signal. Frequency by class shows where the automation's envelope actually ends; severity by class shows where the risk lives; recurrence by class shows what upstream fix would pay. The taxonomy also drives the staffing model — a policy-conflict class needs someone with policy authority, a data-conflict class needs someone who can bind the source systems — and it distinguishes the routine-recoverable from the containment-now cases, which no first-in-first-out mailbox can do.

Build the taxonomy from history, not imagination: sample the last few hundred exceptions from the queue as it actually exists, have the people who resolved them name the reasons in their own words, and expect eight to twelve classes to cover nearly everything. The residual 'novel' class is not a failure of the taxonomy; it is its most important member, because novel cases are where tomorrow's classes announce themselves.

The taxonomy also protects the operation from a subtle staffing error: treating exception volume as one number. A hundred weekly exceptions that are ninety percent routine-data and ten percent policy-conflict need one clerk-hour profile; the same hundred at reversed proportions need policy authority on call. Classification is what makes the difference visible before the staffing model discovers it through backlog.

Detect and contain

The second part is the system's obligation to know when it is out of its depth — and to stop moving when it is. Detection is boundary awareness: input validation, confidence thresholds, policy checks, dependency health, and anomaly signals, each operationalized rather than aspirational. Containment is the safe state that follows: hold the case, release nothing partial, mark what was attempted, and surface the event while response options are still open.

The failure this part prevents is the plausible guess — the system that, rather than declaring an exception it was designed to declare, proceeds instead with its own best interpretation of a malformed input, a conflicting record, or a silently failed dependency. Plausible guesses are the most expensive failure class in automation because they convert detectable exceptions into undetected errors, which surface later, downstream, stripped of the context that

would have made them cheap to fix. A workflow that never raises exceptions is not robust; it is either lucky or quietly guessing, and the difference is discovered at audit.

Partial completion deserves specific design attention: an automation that executes three steps of five before failing has created a state no one designed. Containment means either transactional behavior — all steps or none — or explicit checkpoint states with known recovery procedures. The evidence for this part working is measurable: detection rates on seeded bad inputs, false-accept and false-reject rates at the boundary, time from boundary event to contained state, and the count of partial actions found later — a number whose target is zero and whose actual value is the honesty test.

Route with context

The third part delivers the case to a human as a decision package, not an alarm. The reviewer needs what the system knew — the input, the processing attempted, the reason processing stopped, the affected records and parties, and the clock on the case — assembled at handoff, not reconstructed by the reviewer across four systems and two chat threads. Context reconstruction is the dominant cost in most exception handling, and it is pure waste: the system had the context and dropped it.

Routing itself follows the taxonomy: each class maps to a role with the authority and competence to resolve it, with a fallback when that role is unavailable and an escalation clock for cases that exceed the first responder. The anti-pattern is broadcast routing — the exception channel everyone can see and no one owns — which produces the bystander queue: visible to twelve people, worked by none, aging politely until someone important asks about a specific case.

The measures are queue mechanics: complete-handoff rate (could the reviewer act without hunting?), time to acceptance, clarification round-trips, misroutes, and queue age by class. Age by class is the executive view — a routine-data class aging a week is an efficiency problem; a suspected-harm class aging a day is an incident in progress.

Routing design should also respect the reviewer's clock explicitly. Every routed case carries two times: how long it may wait for acceptance, and how long resolution may take once accepted — both set by class consequence, both visible, both escalating automatically when breached. The clocks convert 'someone will get to it' into a managed commitment, and their breach data is how the operation learns which classes are understaffed before the aging tail teaches it publicly.

Assign response ownership

The fourth part separates two jobs that the mailbox blurred into one: resolving this case, and fixing the condition that keeps producing cases like it. Case ownership belongs in the operation — someone answers for each item reaching resolution inside a service expectation. Condition ownership belongs wherever the cause lives: the product team whose parser drops the field, the data owner whose feed conflicts, the policy owner whose rule contradicts practice.

Without the second owner, the operation clears cases forever. The queue becomes a permanent institution with headcount, dashboards, and no mandate to shrink — the same hundred-case-a-week class resolved heroically, indefinitely, while its cause sits untouched in a backlog nobody reviews. The design is a standing loop: recurrence

reports by class flow to condition owners, remediation commitments come back with dates, and classes that persist past their commitments escalate as risk acceptances someone must sign.

The signature matters. An exception class that recurs because fixing it is not worth the cost is a legitimate business decision — priced, owned, and revisited. The same class recurring because no one with authority ever looked is an unpriced liability that happens to sit in an operations queue. The ownership design exists to make the two distinguishable.

Recover to a known state

The fifth part defines what 'resolved' means, and it means more than 'ticket closed.' Recovery restores the world: data corrected in every system the case touched, partial transactions unwound or completed, downstream consumers notified where they consumed wrong outputs, customer commitments repaired, and the workflow returned to a state from which ordinary processing can safely resume. A case closed by workaround — the record fixed in one system, the discrepancy left in two others — is not resolved; it is deferred with interest.

Recovery procedures are per-class and rehearsed: the rollback for a partial payment run, the reconciliation for a duplicate-record class, the re-entry check before a contained case rejoins the flow. Rehearsal is what separates a procedure from a document — the first live execution of an unrehearsed rollback is itself an incident, performed under pressure, on real data. The evidence: recovery time by class, reconciliation results, repeat-contact rates (the customer calling again is recovery failing in public), and residual-inconsistency findings from periodic cross-system sampling.

The re-entry check closes the loop on containment: before a resolved case rejoins ordinary processing, something verifies the condition that ejected it is actually addressed. Cases that bounce — exception, resolution, re-entry, same exception — are the signature of workaround-as-resolution, and their rate belongs on the same page as the recovery times.

Evidence preservation runs through recovery and beyond it: what the system attempted, what the reviewer decided, and why, kept with the case. The record serves three masters at once — the audit that will eventually ask, the recurrence analysis that needs resolved cases as raw material, and the next reviewer of a similar case, who inherits reasoning instead of reinventing it. Evidence that evaporates at ticket-close forces every future instance of the class to start from zero.

Learn and redesign

The sixth part converts the exception stream into the workflow's improvement budget. Every class is a hypothesis about what to fix: input validation that would retire a malformed-data class, a source-system change that would end a conflict class, a policy clarification that would dissolve a policy-conflict class, an envelope expansion that evidence now supports — or an envelope contraction that evidence now demands. The queue, read quarterly, is the most honest product-management document the automation has.

The part's characteristic corruption is threshold gaming: manual-review volume becomes a target, and the cheap way to hit it is raising confidence thresholds or reclassifying exceptions as acceptable outputs — declaring victory by redefining defeat. The defense is measuring causes retired rather than volume suppressed: a class that

disappeared because validation now catches it upstream is a win; a class that disappeared because the threshold moved is a risk transfer to whoever meets the errors downstream.

Learning also feeds the evaluation machinery: resolved exceptions are pre-labeled hard cases, and the interesting ones belong in the workflow's regression set so the next model or prompt change is tested against the operation's accumulated difficulty. An automation program that discards its exception history is throwing away the only test data that arrives pre-validated by consequence.

AI-based automation widens the taxonomy in one specific way worth naming: the low-confidence and suspected-harm classes stop being rare. A deterministic parser fails on malformed input; a generative system can also fail on well-formed input it merely handles badly, with output that looks complete. Confidence thresholds, sampling of accepted outputs, and downstream verification signals become detection instruments in their own right, and the exception path inherits cases where 'what went wrong' is a judgment rather than an error code. The six parts hold unchanged; the detection investment and reviewer skill profile shift toward evaluation [2].

The path as the readiness standard

Run the six parts together and the opening inversion becomes operational: a workflow is automated when its exception path is designed, staffed, instrumented, and rehearsed — not when its happy path demonstrates throughput. The readiness review is short and concrete: show the taxonomy, show the containment behavior on seeded bad cases, show a routed case's context package, name the case and condition owners, walk the rehearsed recovery for the worst class, and show last quarter's retired causes. A team that can do those six things has an automated workflow. A team that cannot has a demo with an inbox.

EXHIBIT 2

The obligation each part makes inspectable

Classify exceptions

Separate routine recoverable cases from events requiring immediate containment or policy decision.

Detect and contain

Stop the case—and sometimes the workflow—when detection or containment is unreliable.

Route with context

Redesign routing when review effort is dominated by reconstructing context.

Assign response ownership

Reduce automation scope when systemic exceptions persist without accountable remediation.

Recover to a known state

Keep actions reversible until recovery is demonstrated under representative conditions.

Learn and redesign

Expand the happy path only when the exception path shows the operation can carry the additional reliance.

The standard also scales the right way. A low-consequence workflow earns a light path — few classes, simple containment, one owner, modest instrumentation. A workflow touching money, customers, or regulated records earns the full design. What no workflow earns is the sentence 'exceptions route to manual review' standing in for all six parts, because that sentence, unexamined, is where the twelve-hundred-item mailbox came from.

Building the path for a live workflow

The build order follows the evidence:

- Build the taxonomy from real historical cases and frontline interviews.
- Design detection, containment, routing, ownership, evidence, and recovery for each high-consequence class.
- Exercise the path with missing, ambiguous, conflicting, and unsafe inputs before launch.
- Review exception demand as a capacity, product, and operations signal.

Pre-launch exercising is where the path proves it exists. Feed the workflow its monsters deliberately — missing fields, conflicting records, ambiguous categories, an unauthorized request, an unsafe output — and watch each one classify, contain, route, and recover end to end. The exercise costs a day and finds the gap between the path as drawn and the path as built, which every first exercise does. A launch gate that requires the exercise report has, in one requirement, made 'exceptions route to manual review' an impossible sentence to ship.

For workflows already live — which is most of them — the sequence starts with the archaeology: open the existing queue, classify a sample, and triage by severity before designing anything. The five-month mailbox yielded its two regulatory notices in the first afternoon of classification; the taxonomy could wait a week, but those two items could not. Expect the initial classification to reprice the whole automation program's risk posture, usually upward, occasionally dramatically.

Volume forecasting for the path is part of every scale decision, and the arithmetic is unforgiving: doubling happy-path volume roughly doubles exception volume at the current envelope, and the specialist pool does not double by wishing. A scale proposal should state the expected exception load by class, the staffing that absorbs it, and the retirement work that will narrow it — or concede that the plan is to let the queue age. Making that concession explicit has stopped more premature scale-ups than any model metric.

Staff the path as a designed role, not a punishment posting. Exception handling done well is senior work — judgment on the hardest cases, pattern recognition across them, authority to bind systems and policies — and organizations that treat it as overflow labor get the queue quality they staffed for. The quarterly exception review, chaired by the workflow owner with condition owners present, is where the role's findings become the program's roadmap.

What the record should show

A working path leaves inspection-ready evidence:

- A taxonomy with frequency, severity, and recurrence by class — current, not archival.
- Containment results on seeded inputs, and partial-action findings trending to zero.

- Context-complete handoffs and queue age by class, with escalation clocks honored.
- Case and condition owners named per class, with remediation commitments dated.
- Recovery rehearsals completed, and repeat-contact and re-entry-bounce rates.
- Causes retired per quarter — the number that shows the path shrinking itself.

Read the record quarterly with two questions: where is the path absorbing consequence well, and where is it subsidizing a defect that should be fixed at the source? The first question defends the staffing; the second directs the roadmap. A review that asks only the first builds a better and better buffer around an unimproved machine — competent, expensive, and permanent.

The last number is the health of the whole design. A mature exception path gets quieter over time in the classes it has learned and louder only at the frontier of new work — new case types, new volumes, new models. A path whose volume is flat across quarters with no retired causes is an operations team subsidizing an unimproved automation, however smoothly the subsidy runs.

The strongest counterargument

Not every rare event deserves bespoke automation. Some exceptions should remain human-owned, and some cases should be excluded entirely. The objective is not zero exceptions; it is a deliberate path that contains consequence and makes learning possible.

The investment objection — six-part design as gold-plating for a queue that three people were handling fine — should be tested against the queue itself, and the test is usually decisive. Classify one sample of the existing pile and price what surfaces: the aged compliance items, the specialist hours lost to context reconstruction, the workaround resolutions accruing reconciliation debt, the repeat contacts. In my experience the sample pays for the design several times over before the design starts. Where it truly does not — genuinely low consequence, genuinely low volume — the framework itself says so: earn a light path, and spend the rigor where the queue is expensive.

The shift

The shift is definitional: automated means the failure modes are governed, not merely that the success mode is fast. Design the path for the cases that fall outside the rules — classified by cause, contained before consequence spreads, routed with their context intact, owned twice, recovered fully across every touched system, and mined quarterly for the causes worth retiring — and the happy path can then be trusted with everything else, because everything else is what it was actually built for.

The readiness standard travels well, too: it works as a due-diligence question for vendor automations, as an audit lens for inherited workflows, and as a one-line policy for new builds — no launch without the exercised path.

The mailbox is gone now. Its successor is a classified queue with owners, clocks, and a quarterly review that has retired eleven root causes in eighteen months — and the automation it guards now covers more volume than it did, because every retired cause widened the envelope the operation could actually stand behind. That is the relationship the design creates: the exception path is not the tax on automation. It is how automation earns its next percent.

Notes and sources

[1] NIST, "AI Risk Management Framework Core," 2023. https://airc.nist.gov/airmf-resources/airmf/5-sec-core/

[2] Blake Bullwinkel et al., "Lessons From Red Teaming 100 Generative AI Products," Microsoft Research, January 2025. https://www.microsoft.com/en-us/research/publication/lessons-from-red-teaming-100-generative-ai-products/

About the author

Marco Policani is an enterprise portfolio, PMO, and AI operating-governance leader. He builds portfolio governance systems where AI carries the analytical load and named humans own every decision — an approach documented across case studies, walkthroughs, and working governance guides on his portfolio site.

Portfolio & governance library: policani.net/governance · Full portfolio: policani.net · LinkedIn: linkedin.com/in/marcpolicani