From RPA Rules to AI Recursion

Replacing robotic process automation with self-improving AI agents, and why recursive feedback and the DMAIC cycle are the cornerstones that make the substitution compound rather than drift.

Abstract

Robotic process automation (RPA) automates a task by recording a fixed sequence of clicks and rules against interfaces it does not understand. It works until the world moves: a renamed field, a new exception, a changed form, and the bot breaks or, worse, carries on wrongly. Because the rules are frozen and the environment is not, an RPA estate decays, and its promised savings are eaten by a rising maintenance bill. We argue that AI agents substitute for RPA not because they execute the same rules faster but because they close a loop RPA cannot: they interpret intent, and they *improve the automation from its own execution*, cycle after cycle. Recursion is the first cornerstone. But a self-improving loop left to itself drifts, optimises the wrong thing, or silently regresses, so the second cornerstone is the discipline that makes the loop converge and stay accountable: the Six Sigma *DMAIC* cycle, Define, Measure, Analyse, Improve, Control, read as the control structure of the automation rather than a one-off project. We give a simple model in which an RPA bot's error rate drifts upward between manual fixes while a DMAIC-governed agent's error rate falls geometrically to an irreducible floor, and we derive a *substitution condition*: the agent replaces the bot when its loop removes error faster than the environment injects it, a single dimensionless number. The Control phase is what makes this safe, a non-regression gate that only commits a change that measurably does not make things worse, turning improvement into a ratchet. A constructed accounts-payable case traces the numbers. This is practitioner-facing argument, not a benchmark.

Keywords: robotic process automation; RPA; intelligent automation; AI agents; agentic automation; recursive self-improvement; feedback loops; DMAIC; Six Sigma; statistical process control; continuous improvement; process mining; straight-through processing; automation maintenance; control theory; requisite variety

1. Introduction

Robotic process automation made a modest promise and, for a time, kept it. Where a person moved data between systems that could not talk to each other, logging into a portal, copying figures into a spreadsheet, keying them into an ERP, an RPA *bot* could be taught to do the same, clicking the same buttons in the same order, faster and without tiring. The appeal was that it needed no change to the underlying systems: the bot drove the existing screens exactly as a person did, so a department could automate a process in weeks without waiting for IT to build an interface. On that basis RPA became one of the fastest-adopted enterprise technologies of the last decade.

The trouble is in the word *robotic*. An RPA bot does not understand the task; it repeats a recording. It knows that the invoice total is in the third box on the second tab, not what an invoice total is, so when the portal is redesigned and the box moves, the bot does not adapt, it fails, and if it fails silently it keeps posting the wrong number until someone notices. Every process an organisation runs sits in an environment that drifts: software is updated, forms are redesigned, suppliers send a new layout, a regulation adds a field, an edge case arrives that the recording never saw. Against a drifting environment a frozen rule set is a wasting asset. The industry name for the consequence is *bot maintenance*, and it is the quiet scandal of the field: a large share of the effort in a mature RPA programme goes not into automating new work but into repairing automations that the world has broken, so that the running cost rises over time and the business case erodes from under it.

This paper is about what replaces RPA, and why the replacement is a different kind of thing rather than a faster bot. A capable AI agent, a model that can read a screen or a document, reason about what it is for, call tools and check its own work, can do the clerical task RPA did, but that is not the interesting claim, because doing the same brittle thing with a more expensive engine is no advance. The interesting claim is that an agent can do what a bot structurally cannot: it can *improve the automation from its own running*. Each time it handles a case it generates evidence, what it did, whether the result was accepted, where it hesitated or was corrected, and that evidence can be fed back to revise how the automation behaves next time. Automation stops being a recording made once and decaying thereafter and becomes a loop that gets better the more it runs. We call this *recursive* automation, and it is the first of the paper's two cornerstones.

The second cornerstone is the one that makes the first safe, and without it the recursion is a liability rather than an asset. A loop that changes its own behaviour from its own outputs is, in control terms, a feedback system, and feedback systems can converge or they can run away. An agent left to tune itself against a noisy signal will chase the noise, optimise the measure it was given rather than the outcome it was meant to serve, and regress in ways no one chose. The discipline that converts a self-improving loop from a danger into a dependable engine of improvement is not new; industry has had it since the 1920s and formalised it for the last thirty years as *Six Sigma*, whose operational core is the *DMAIC* cycle: Define the goal and the tolerances, Measure what actually happens, Analyse the sources of error and variation, Improve by a deliberate change, and Control so that the gain is held and not lost.

Read as a project methodology, DMAIC is familiar. Read as the control structure of a self-improving automation, run continuously rather than once, it is exactly the governor the recursive loop needs, and the pairing is the subject of this paper.

The contribution is an argument and a small model, not a benchmark. Section 2 places the claim against the record of RPA, intelligent automation, continuous improvement and AI agents. Section 3 sets out why a fixed-rule bot decays, with a model of drift and maintenance. Section 4 describes the recursive loop that an agent makes possible. Section 5 is the heart of the paper: DMAIC as the control structure that makes the loop converge, with the Control phase as a non-regression ratchet. Section 6 derives the substitution condition, a single number that says when the agent overtakes the bot. Section 7 traces a constructed accounts-payable case; Sections 8 to 11 draw the consequences for how an automation function is run, answer objections, state limits and conclude. The question throughout is: **once automation can learn from its own execution, what keeps it learning the right thing, and when does that make the recorded bot obsolete?**

2. Background

2.1. RPA and its ceiling

RPA was characterised in the information-systems literature as "lightweight" automation: software that operates the presentation layer of existing systems rather than their data or logic, so that it is deployed by the business, outside the traditional IT project, against unchanged back ends (van der Aalst, Bichler & Heinzl, 2018; Hofmann, Samp & Urbach, 2020). The early case evidence was genuinely positive on cost and speed (Lacity & Willcocks, 2016; Willcocks, Lacity & Craig, 2015). The same literature, as the field matured, named the ceiling. Syed et al. (2020), reviewing the state of RPA research and practice, identify the dependence on stable, structured inputs and the fragility of bots to interface and process change as recurring, unresolved themes, and locate maintenance and governance among the field's central challenges; Wewerka and Reichert (2023), mapping the field systematically, catalogue limited exception handling and maintenance effort among its persistent problems. The ceiling is not a defect of a particular product; it is intrinsic to automating by imitation of the surface. A bot encodes *how* a competent person happened to click, not *what* the task requires, and the gap between the two is exactly the room the environment has to break it.

2.2. From RPA to intelligent and agentic automation

Practitioners responded by bolting intelligence onto the bot: optical character recognition to read documents, a classifier to sort them, a model to handle the unstructured input RPA could not. This *intelligent automation* or *hyperautomation* widened the range of inputs a pipeline could accept, but it left the control structure unchanged, a fixed orchestration with smarter components wired into fixed positions, and so it inherited the same brittleness one layer up. The more recent shift is of a different kind. An AI *agent* does not occupy a fixed position in a hand-drawn flow; it is given a goal and a set of tools and decides the steps, which lets it absorb variation that would have broken a fixed orchestration, and, because it reasons over the task rather than replaying a recording, it can be asked to explain, check and revise what it did. The agent literature in machine learning gives the mechanics, interleaving reasoning with tool use (Yao et al., 2023), reflecting on failed attempts to do better on the next (Shinn et al., 2023), and the broader survey of autonomous agents (Wang et al., 2024). Our interest is not the mechanism in the abstract but a consequence for operations: an agent's execution produces inspectable evidence, and evidence is what a loop needs to improve.

2.3. Continuous improvement is older than the machine

The idea that a process should be measured and improved from its own output is not an AI idea; it is the foundation of twentieth-century quality management, and the paper leans on it deliberately. Shewhart (1931) distinguished the variation inherent in a stable process, *common cause*, from the assignable, *special cause*, variation that signals something has changed, and gave the control chart as the instrument that tells them apart, so that one reacts to real shifts and not to noise, the exact failure mode of an ungoverned learning loop; the modern statistical treatment is Montgomery (2024). Deming (1986) built the management system around it and popularised the Plan–Do–Study–Act cycle of iterative improvement. Six Sigma (Harry & Schroeder, 2000; Pyzdek & Keller, 2023) packaged the practice into DMAIC and tied it to a quantified defect rate; the academic treatments give it a theory (Linderman et al., 2003) and a candid account of its tension with innovation (Benner & Tushman, 2003). One strand is especially close to this paper: Box's (1957) *evolutionary operation*, which ran a production process as a slow, continual experiment, nudging its settings and keeping what measurably improved yield, is recursive self-improvement under statistical control, decades before the term, and it is the right ancestor for what an agent now does at the speed of software.

2.4. Feedback, variety and the limits of a fixed rule

Cybernetics supplies the reason a fixed bot must lose to a varying world. Ashby's (1956) law of requisite variety holds that a regulator can absorb disturbance only in proportion to the variety it can itself produce: only variety can destroy variety. An RPA bot has, by construction, almost none, one response per recorded path, so it can regulate only an environment that never leaves those paths, and any real environment eventually does. An agent governed by a learning loop manufactures variety on demand, generating a response to a case it was not explicitly built for, which is why it can track a moving target that a fixed rule cannot. Wiener (1948) gave the founding account of control by feedback, and the machine-learning tradition of learning from a reward signal (Sutton & Barto, 2018) and of organisations learning by correcting error (Argyris & Schön, 1978) are the same principle at other scales. The contribution here is to put these two old bodies of knowledge together on a new object: the agent supplies the variety, and DMAIC supplies the control that keeps the variety pointed at the goal.

3. Why a fixed-rule bot decays

The case against RPA is not that it fails on day one; it is that it is built to be optimal on day one and to degrade every day

after. Fix a process and deploy a bot that handles a share of its cases correctly. The environment then drifts, at some rate, away from the state the bot was recorded against: forms change, systems are upgraded, new variants of inputs arrive, policies add a step. Each increment of drift turns some cases the bot used to handle into cases it now mishandles, so the bot's error rate rises roughly in proportion to the accumulated drift. Between interventions,

$$\varepsilon^{\mathrm{rpa}}(t) = \varepsilon_0 + \delta\, t, \tag{1}$$

A fixed bot's error rate rises with accumulated environmental drift at rate δ.

where ε_0 is the error rate at deployment, t is time, and δ is the drift rate, the pace at which the world leaves the recording behind. The only thing that lowers $\varepsilon^{\mathrm{rpa}}$ is a person: an engineer notices the breakage, re-records the path, and resets the error rate toward ε_0, at a cost, and until the next drift. The bot therefore traces a sawtooth, climbing with drift and dropping at each manual repair, and never improving on its original design, because nothing in it learns. Its lifetime average error is set by how fast the world moves (δ) and how often the organisation can afford to send someone to fix it.

Two features of this picture matter. First, the maintenance is *reactive and unbounded*: as an estate of bots grows, the drift they collectively accumulate grows with it, so maintenance effort rises with the number of automations rather than with the new work being automated, which is the mechanism by which an RPA programme's running cost overtakes its savings. Second, the failure is often *silent*: a bot that posts a value from the wrong field does not throw an error, it produces a plausible, wrong result, and the cost of that is bounded only by how long it goes unnoticed. A fixed rule in a moving world is not merely limited; it is a liability whose size the organisation does not control.

4. Recursive automation: the loop a bot cannot close

An agent changes the picture because its execution is a source of information that can be turned back on the automation itself. When an agent processes a case it leaves a trace: the input it saw, the steps it chose, the tool calls it made, the output it produced, and, crucially, the downstream signal of whether that output was accepted, corrected or reversed. That trace is evidence about where the automation is strong and where it is weak, and because the agent's behaviour is expressed in things that can be edited, its instructions, the examples it is given, the tools and checks it must use, the routing of hard cases, that evidence can be fed back as a revision. The automation improves from its own running. Where equation (1) has only an external hand to lower the error, the recursive loop lowers it from the inside, each cycle:

$$\varepsilon_{k+1} = \varepsilon_* + \rho\,(\varepsilon_k - \varepsilon_*), \qquad \rho = 1 - g, \tag{2}$$

Each improvement cycle removes a fraction g of the error above the floor ε_*.

Here ε_k is the error rate after k cycles, $g \in (0,1)$ is the *improvement gain*, the fraction of the remediable error a cycle removes, and ε_* is the *irreducible floor*, the residual the process cannot shed because it reflects genuine ambiguity, missing information or irreducible judgement, the essential rather than accidental difficulty of the task. Iterating equation (2) gives the closed form

$$\varepsilon_k = \varepsilon_* + (\varepsilon_0 - \varepsilon_*)\,\rho^{\,k}, \tag{3}$$

Error falls geometrically from its starting value to the floor as cycles accumulate.

a geometric fall from the starting error to the floor. This is the shape the quality tradition has always aimed for and the shape a bot can never trace, because a bot has no g: its improvement gain is zero, and equation (2) collapses to a constant interrupted only by manual resets. The difference between RPA and agentic automation, in this model, is the difference between $g = 0$ and $g > 0$: between an automation that can only be repaired from outside and one that improves itself from within.

Three conditions make g positive and keep it so, and all three are things the agent's execution supplies that a bot's does not. There must be a *signal*: a measurable outcome for each case, so that better and worse can be told apart. There must be *attributable evidence*: traces rich enough to locate why a case went wrong, not merely that it did. And there must be an *editable surface*: levers, prompts, examples, tools, routing, that a revision can actually move. An agent produces all three as a by-product of working; a recorded bot produces none of them, which is the deeper reason it cannot learn. But a positive g is not yet a safe g. A loop that revises itself against a signal can just as easily raise the error as lower it, if the signal is noisy, the attribution wrong, or the change a regression. Making g reliably positive, cycle after cycle, is the job of the control structure, and that is DMAIC.

5. DMAIC as the control structure

The paper's central claim is that the discipline which turns a self-improving loop from a hazard into a dependable engine already exists, fully worked out, in the Six Sigma DMAIC cycle, provided it is read not as a one-off improvement project but as the *standing control structure* of the automation, run continuously. Each phase supplies something the recursive loop of Section 4 needs to keep its gain positive and its direction true; Figure 1 draws the cycle and Table 3 maps each phase to its role.

Define. The loop needs a goal that is not the proxy it will be tempted to chase. Define states the outcome the automation serves, the tolerances that count as success, and an honest estimate of the floor ε_* below which further effort is waste, so that the loop optimises the business result and not a convenient surrogate for it. Most runaway feedback failures are Define failures: the loop improved exactly what it was told to, which was the wrong thing.

Measure. There is no improvement gain without a signal, and Measure is the phase that manufactures one, instrumenting every case so that the outcome, the corrections

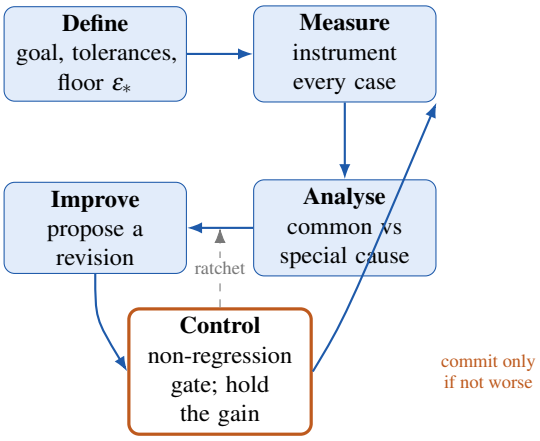


Figure 1. DMAIC read as the standing control structure of a recursive automation. Define fixes the goal and the floor; Measure instruments every case; Analyse separates real shifts from noise; Improve proposes a revision; Control is a non-regression gate that commits a change only when it measurably does not worsen the outcome, then holds the gain. The gate (orange) is what makes the loop a ratchet rather than a random walk: improvement accumulates and is not given back.

and the points of hesitation are recorded, not sampled by anecdote. This is the modern form of Shewhart's insight that you cannot improve what you do not measure, and it is also where process mining (van der Aalst, 2016) earns its place, reconstructing what the process actually did from its own event logs rather than from how it was imagined to run.

Analyse. A learning loop's characteristic error is to react to noise as though it were signal, re-tuning itself against the normal scatter of a stable process and thrashing. Analyse is the guard: it separates common-cause variation, to be left alone, from special-cause variation, to be acted on, and it attributes the actionable error to a cause specific enough to fix. This is Shewhart's (1931) distinction doing exactly the work it was invented for, now protecting an automation from chasing its own tail.

Improve. Only now is a change made: a revision to the agent's instructions, its examples, its tools, its checks, or the routing that sends a class of hard case to a human or a specialised path. Improve is the mutation step of the loop, the place the gain g is actually realised, and it is deliberately downstream of measurement and analysis so that the change answers a diagnosed cause rather than a hunch.

Control. Control is the phase that makes the whole thing safe, and it is where the recursive automation differs most sharply from both a bot and an ungoverned learning system. A proposed change is not adopted because it seems better; it is adopted only if it is measured, against held-out cases, not to make the outcome worse, and otherwise it is rejected and the previous version retained. Formally, writing $\widehat{\varepsilon}(m)$ for the measured error of a candidate revision m,

$$\text{commit } m \iff \widehat{\varepsilon}(m) \le \varepsilon_k - \gamma, \tag{4}$$

The Control gate: adopt a revision only when it measurably beats the current version by a margin γ.

where $\gamma \ge 0$ is a confidence margin that keeps the gate from being fooled by noise. The gate has a precise effect on the loop: it makes the error sequence monotone in expectation,

$$\mathbb{E}[\varepsilon_{k+1} \mid \varepsilon_k] \le \varepsilon_k, \tag{5}$$

Under the Control gate the error can only hold or fall, never climb: improvement is a ratchet.

so improvement accumulates and is never given back. This is what converts the raw recursion of equation (2) from a random walk that might drift anywhere into a ratchet that only tightens, and it is the discipline Statistical Process Control contributed to manufacturing a century ago, now applied to an automation that rewrites itself. Control also does the unglamorous other half of its job: it standardises and holds the gain, pinning the improved version in place with its guardrails so that the next cycle starts from the new floor and not the old one.

The pairing is the point. Recursion without DMAIC is a loop that can improve or wreck with equal ease; DMAIC without recursion is the periodic improvement project that quality teams have always run by hand, bounded by how often people can convene. An agent that runs DMAIC as its own standing control structure gets both: the speed and reach of software improvement and the convergence guarantee of statistical control.

6. The substitution condition

We can now say precisely when the agent overtakes the bot. The two automations face the same drifting world, at the same drift rate δ. The bot has no internal gain and so, by equation (1), its error climbs with drift until a person resets it. The agent runs its DMAIC loop with cycle time τ, the time to measure, analyse, propose and gate one revision, and per-cycle gain g, while the same drift δ pushes its error up. Over continuous time the two forces combine:

$$\dot{\varepsilon} = -\frac{g}{\tau}(\varepsilon - \varepsilon_*) + \delta, \tag{6}$$

The agent's error falls under its own loop at rate g/τ and rises under drift at rate δ.

a learning term pulling the error down toward the floor and a drift term pushing it up. Its steady state, where the loop exactly keeps pace with the world, is

$$\varepsilon_\infty = \varepsilon_* + \frac{\delta\,\tau}{g}, \tag{7}$$

The controlled automation settles a margin $\delta\tau/g$ above the irreducible floor.

so the automation holds at a bounded distance above the floor, set by the ratio of how fast the world drifts to how fast the loop learns. The bot, with $g \to 0$, has no finite steady state at all: its error is bounded only by the maintenance schedule. Define the dimensionless *substitution number*

$$\Sigma = \frac{g}{\delta\,\tau}, \tag{8}$$

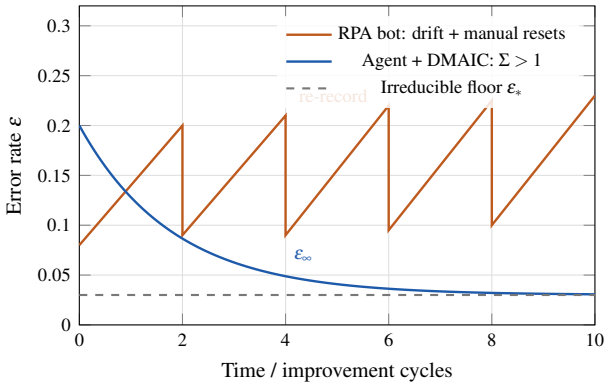


Figure 2. Two automations in the same drifting environment. The RPA bot (orange) climbs with drift and is reset only by manual re-recording, tracing a sawtooth whose average never improves on day one. The DMAIC-governed agent (blue) removes error faster than drift adds it ($\Sigma > 1$) and converges geometrically to a tight band above the irreducible floor ε_* (dashed). Values illustrative.

The substitution number: improvement gain per cycle against drift accumulated per cycle.

the ratio of the error a cycle removes to the error drift adds over that same cycle. When $\Sigma > 1$ the loop removes error faster than the environment injects it and the automation converges to a tight band above the floor; when $\Sigma \gg 1$ it sits essentially at the floor. The agent substitutes for the bot precisely when it can run a governed loop with $\Sigma > 1$, which is a statement about three controllable quantities: raise the gain g with better measurement and attribution, shorten the cycle τ with automation of the loop itself, and the agent beats any fixed bot in any environment whose drift δ is finite. Figure 2 contrasts the two trajectories. The practical reading is that substitution is not a question of whether the agent is cleverer than the bot on day one, it often is not, but of whether its loop turns, under control, faster than the world turns; and once it does, the bot's sawtooth and the agent's descent diverge for good.

7. A constructed case: accounts payable

The case is constructed, not observed: its figures are illustrative and internally consistent, and it is meant to show the model's quantities taking plausible values, not to report a result. Consider the accounts-payable function of a mid-sized group that receives supplier invoices in many formats across several portals and must match each to a purchase order and a goods receipt, code it, and post it for payment. It is the archetypal RPA target, high-volume, rule-describable, boring, and the archetypal RPA disappointment, because the formats drift constantly and every supplier is an exception waiting to happen.

The group's RPA estate handles the cleanest supplier layouts and routes everything else to people. At deployment it posted about 55 per cent of invoices straight through, but over eighteen months three portal updates and a wave of new suppliers pushed the straight-through rate down and the silent-error rate up, so that a standing team spends much of its week re-recording broken bots and chasing mispostings, Table 1 gives the before column. Replacing the recorded bots with an agent that reads each invoice for what it is,

Table 1. Accounts payable, before and after substitution. The RPA estate decays between manual repairs; the DMAIC-governed agent converges toward the floor set by invoices needing genuine judgement. Illustrative figures for the constructed group.

	RPA estate	Agent + DMAIC
Straight-through rate at start	55%	60%
Straight-through rate at 12 months	44%	88%
Silent-error rate (mispostings)	rising	falling, gated
Improvement gain per cycle g	0	≈ 0.4
Loop cycle time τ	n/a	1 week
Substitution number Σ	0	> 1
Dominant human work	re-recording bots	reviewing gated chang
Running cost over time	rising	falling to a floor

matches it by reasoning rather than by fixed coordinates, and routes genuine exceptions to a person, and wrapping that agent in a standing DMAIC loop, the straight-through rate is the thing Measure tracks, each week's corrections are the evidence Analyse works on, the agent's prompts, examples and matching checks are what Improve revises, and a held-out sample is the gate Control applies, changes both the level and its direction. The agent starts only a little better than the bot but, because its loop turns faster than the portals drift ($\Sigma > 1$), its exception rate falls cycle on cycle toward the floor set by the invoices that genuinely need human judgement, while the maintenance burden inverts: instead of engineers repairing recordings, the work becomes reviewing the loop's proposed changes at the Control gate. Table 1 traces the quantities of the model through the case.

8. What changes for the operating model

Substituting an agent for a bot is not a like-for-like swap of engines under the same bonnet; it changes what the automation function does and what it must be good at. Table 2 sets the two side by side. The largest change is where the human effort goes. Under RPA the recurring human work is *repair*: finding and re-recording the bots the world has broken, a reactive task that scales with the size of the estate and produces nothing new. Under a DMAIC-governed agent the recurring human work is *the Control gate*: reviewing the loop's proposed revisions and deciding what to commit, a task that scales with the *rate of improvement* rather than the size of the estate and that produces a better automation each time it is done. The skills change accordingly, from scripting robots against screens to defining outcomes, instrumenting processes and judging evidence, which is to say from automation engineering toward the discipline of quality management, now carried out at the speed of software.

There is a second change, in governance. A system that rewrites its own behaviour is more powerful than a recording and demands more accountability, not less, and DMAIC supplies the audit trail as a by-product: every committed change answers a measured cause, passed a recorded gate, and can be attributed and reverted, so that a self-improving automation is, properly run, more inspectable than the opaque bot it replaces, not less. The Control phase is as much a compliance instrument as a quality one. This matters most where automation touches regulated work, and it is the same

Table 2. Two kinds of automation. RPA imitates the surface of a task and decays; a DMAIC-governed agent reasons about the task and improves. The control structure, not the engine, is the real difference.

	RPA bot	**Agent + DMAIC**
Automates	recorded clicks and rules	the intent of the task
Handles variation	only on recorded paths	generates a response
On drift	breaks or errs silently	adapts, then learns
Improves itself	no ($g = 0$)	yes, under control
Recurring human work	repair the estate	review the gate
Cost over time	rises with estate	falls to a floor
Failure mode	frozen rule, moving world	chasing a bad signal
Guard against it	manual maintenance	DMAIC control

Table 3. DMAIC read as a standing control structure. Each phase supplies something the recursive loop needs to keep its gain positive and its direction true, and guards a characteristic failure of ungoverned self-improvement.

Phase	**Role in the loop**	**Failure it guards**
Define	fix the outcome, tolerances and floor ε_*	optimising the wrong proxy
Measure	instrument every case; make a signal	improving by anecdote
Analyse	common vs special cause; attribute error	chasing noise
Improve	revise prompts, tools, checks, routing	changing on a hunch
Control	non-regression gate; hold the gain	silent regression; drift

discipline, measured change behind a gate, that we have argued for elsewhere in our research on regulated IT.

9. Objections and replies

An agent is far more expensive per case than a bot. Per case, today, often yes, and for a narrow, stable, high-volume task whose environment genuinely does not drift, a recorded bot may remain the economical choice, and nothing here says otherwise. The argument bites where the environment drifts, which is most places, because there the bot's true cost is not its per-case price but its maintenance tail and its silent errors, and the agent's loop pays for its higher unit cost by removing both. The comparison to make is lifetime cost against a drifting environment, not unit cost on day one.

This is just RPA with a model bolted on, which the vendors already sell. It is the opposite. Bolting a model into a fixed orchestration, intelligent automation as commonly sold, keeps the brittle control structure and makes its components cleverer, so it raises the ceiling a little and inherits the decay. The claim here is about the control structure itself: replacing the fixed orchestration with a governed loop that rewrites the automation from its own output. The engine is the smaller half of the change; the loop is the larger.

A self-improving system that changes its own behaviour is a risk you cannot govern. It is a risk, and ungoverned it is a serious one, which is the entire reason the paper's second cornerstone is a control discipline and not a model. An agent that retunes itself against a noisy signal with no Analyse phase and no Control gate will indeed drift and regress. The reply is not to forbid the loop but to govern it with the method manufacturing has used to govern self-adjusting processes for a century: measure, separate signal from noise, change against a diagnosed cause, and commit only what a gate proves does not make things worse. A governed loop is more accountable than a bot, not less.

Your model is too simple; real error does not fall geometrically. The model is a deliberate simplification and its constants are not measured; its purpose is to make the structure legible, that a bot has $g = 0$ and a governed agent has $g > 0$, and that substitution turns on whether the loop out-runs the drift. The geometric form is the generic behaviour of a proportional controller against a floor, not a precise empirical law, and Section 10 says so. The substitution number Σ is meant as a way of thinking, not a figure to quote to three places.

10. Limitations

This is a practitioner-facing argument with a deliberately simple model, and its limits should be plain. The model of Sections 3 to 6 is illustrative: the linear drift of equation (1), the constant gain of equation (2) and the single floor ε_* are simplifications chosen for legibility, and real drift is lumpy, real gain diminishes as the floor nears, and the floor itself moves as the task changes; the substitution number Σ is a lens, not a measured quantity, and we have not estimated it against field data. The argument is scoped to processes that drift and that carry a usable outcome signal; a genuinely stationary, high-volume task may still favour a recorded bot, and a process with no measurable outcome starves the loop of the signal on which the whole mechanism depends. The claim is about a control structure, not a product, and the specific agent technology is early and moving fast; a reader should keep the structure, a governed recursive loop beats a frozen rule in a moving world, and re-examine the components as they change. Finally, the gain is only ever as honest as the Measure and Control phases: an organisation that instruments badly, or lets the gate decay into a rubber stamp, will get the recursion's risks without its convergence, which is a reason to take the discipline seriously, not a defect of the idea.

11. Conclusion

Robotic process automation did something useful by recording how a person drives a system and replaying it, but it automated the surface of a task and froze it, and a frozen rule in a moving world decays: its error climbs with drift, its maintenance bill rises with its estate, and its worst failures are silent. AI agents substitute for it not by

replaying the same rules with a costlier engine but by closing a loop a recording cannot, improving the automation from its own execution, cycle after cycle. That recursion is the first cornerstone and, alone, as much hazard as help. The second cornerstone is what makes it dependable: the DMAIC cycle, read not as a one-off project but as the standing control structure of the automation, Define the goal and the floor, Measure every case, Analyse signal from noise, Improve against a diagnosed cause, and Control with a gate that commits only what measurably does not make things worse, so that improvement becomes a ratchet and not a gamble. In the paper's model the difference is the difference between an improvement gain of zero and a gain held reliably positive, and the agent overtakes the bot exactly when its governed loop removes error faster than the world injects it. The engineering that follows is less about the model that reads the invoice than about the loop built around it, and the discipline that loop needs is one industry has owned for a century and can now run at the speed of software. Automate the task, measure what happens, improve against what you measure, and let nothing change the automation that a gate has not shown to be better. A bot is a photograph of a process; a governed agent is the process learning. That is the substitution, and it is the whole of it.

Appendix A. Running DMAIC as an automation's control loop

The practices below are what keep a recursive automation converging rather than drifting. None is novel to AI; each is the quality tradition applied to an automation that rewrites itself.

Table 4. A control-loop checklist. Each practice keeps the improvement gain positive and honest; the right-hand column is the failure it prevents.

Practice	What it prevents
Define the business outcome and tolerances before the metric	The loop optimising a convenient proxy
Instrument every case, not a sample	Improving by anecdote; a blind loop
Separate common-cause from special-cause variation before acting	The loop chasing its own noise
Change against a diagnosed cause, one revision at a time	Untraceable, compounding changes
Gate every revision on held-out cases; commit only if not worse	Silent regression
Standardise and hold each gain before the next cycle	Giving back improvement already won
Keep the floor ε_* honest; route the irreducible to people	Spending effort below the floor
Keep the gate's decisions attributable and reversible	An ungovernable self-rewriting system

References

[1] Argyris, C., & Schön, D. A. (1978). *Organizational learning: A theory of action perspective*. Addison-Wesley. ISBN 978-0-201-00174-3.

[2] Ashby, W. R. (1956). *An introduction to cybernetics*. Chapman & Hall. http://pcp.vub.ac.be/books/IntroCyb.pdf

[3] Benner, M. J., & Tushman, M. L. (2003). Exploitation, exploration, and process management: The productivity dilemma revisited. *Academy of Management Review*, 28(2), 238–256. https://doi.org/10.2307/30040711

[4] Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 6(2), 81–101. https://doi.org/10.2307/2985505

[5] Deming, W. E. (1986). *Out of the crisis*. MIT Center for Advanced Engineering Study. ISBN 978-0-911379-01-0.

[6] Harry, M. J., & Schroeder, R. (2000). *Six Sigma: The breakthrough management strategy revolutionizing the world's top corporations*. Currency/Doubleday. ISBN 978-0-385-49437-3.

[7] Hofmann, P., Samp, C., & Urbach, N. (2020). Robotic process automation. *Electronic Markets*, 30(1), 99–106. https://doi.org/10.1007/s12525-019-00365-8

[8] Lacity, M., & Willcocks, L. P. (2016). Robotic process automation at Telefónica O2. *MIS Quarterly Executive*, 15(1), 21–35. https://aisel.aisnet.org/misqe/vol15/iss1/4/

[9] Linderman, K., Schroeder, R. G., Zaheer, S., & Choo, A. S. (2003). Six Sigma: A goal-theoretic perspective. *Journal of Operations Management*, 21(2), 193–203. https://doi.org/10.1016/S0272-6963(02)00087-6

[10] Montgomery, D. C. (2024). *Introduction to statistical quality control* (8th ed.). Wiley. ISBN 978-1-119-72309-7.

[11] Pyzdek, T., & Keller, P. A. (2023). *The Six Sigma handbook* (6th ed.). McGraw-Hill. ISBN 978-1-26-514399-2.

[12] Shewhart, W. A. (1931). *Economic control of quality of manufactured product*. D. Van Nostrand. ISBN 978-0-87389-076-2 (ASQ reprint).

[13] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*. https://arxiv.org/abs/2303.11366

[14] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press. ISBN 978-0-262-03924-6.

[15] Syed, R., Suriadi, S., Adams, M., Bandara, W., Leemans, S. J. J., Ouyang, C., ter Hofstede, A. H. M., van de Weerd, I., Wynn, M. T., & Reijers, H. A. (2020). Robotic process automation: Contemporary themes and challenges. *Computers in Industry*, 115, 103162. https://doi.org/10.1016/j.compind.2019.103162

[16] van der Aalst, W. M. P. (2016). *Process mining: Data science in action* (2nd ed.). Springer. https://doi.org/10.1007/978-3-662-49851-4

[17] van der Aalst, W. M. P., Bichler, M., & Heinzl, A. (2018). Robotic process automation. *Business & Information Systems Engineering*, 60(4), 269–272. https://doi.org/10.1007/s12599-018-0542-4

[18] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J.-R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 186345. https://doi.org/10.1007/s11704-024-40231-1

[19] Wewerka, J., & Reichert, M. (2023). Robotic process automation: A systematic mapping study and classification framework. *Enterprise Information Systems*, 17(2), 1986862. https://doi.org/10.1080/17517575.2021.1986862

[20] Wiener, N. (1948). *Cybernetics: Or control and communication in the animal and the machine*. MIT Press. ISBN 978-0-262-73009-9.

[21] Willcocks, L. P., Lacity, M., & Craig, A. (2015). *The IT function and robotic process automation* (The Outsourcing Unit Working Research Paper Series). London School of Economics and Political Science, LSE Research Online item 64519. http://eprints.lse.ac.uk/64519/

[22] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR 2023)*. https://arxiv.org/abs/2210.03629