TAME: An Effort-Hours Efficiency Model for Test Automation

Test Automation Modelled Effort — with application to AI-assisted test creation and automated execution.

Hari Gunupudi

Abstract

This paper introduces **TAME (Test Automation Modelled Effort)**, a two-stage model for quantifying the human effort saved when a testing process moves from manual work to AI-assisted test creation and automated execution.

Unlike informal "percent faster" claims, TAME measures effort in effort-hours and separates two stages whose economics differ fundamentally. The framework integrates a one-time creation stage, where AI generation plus human review replaces manual authoring; a recurring execution stage, where automated scripts replace manual re-runs each cycle; an accuracy-bounded savings ceiling for creation; and a volume-independent break-even point for automation.

The model reports human effort only; the faster wall-clock turnaround of an unattended run is treated as a separate throughput benefit and excluded from the effort-hours result. The two stages also differ in kind, which is the basis of the investment case: manual execution is a raw operating expense, paid in full every cycle and leaving no asset behind, whereas automation scripting is a capital-like outlay, incurred once and amortised across every cycle it serves. Break-even is the point at which that amortised investment falls below the recurring manual expense.

Notation

All symbols used in the model. Units are minutes unless noted. The default values are **illustrative** — chosen only to make the worked examples concrete — and are not recommendations; every one should be replaced with the team's own measured figures before the model is used to make a decision (see §14).

Symbol	Meaning	Default / units
Creation inputs		
N	Number of test cases needed	400
k	Scenarios prompted into the AI tool	20
t_{author}	Manual authoring time per case	20 min
t_{setup}	Login and prompt time per scenario	10 min
t_{review}	Review time per AI-generated case	4 min
p	Fraction of AI cases needing rework	30%
t_{correct}	Correction time per reworked case	6 min
m	AI miss rate — cases authored manually	10%
Execution inputs		

Symbol	Meaning	Default / units
a	Automation coverage — share of cases scripted	70%
t_{script}	Scripting time per automated case (one-time)	60 min
t_{exec}	Manual execution time per case, per cycle	12 min
t_{triage}	Triage time per automated case, per cycle	1 min
μ	Scripts needing maintenance per cycle	8%
t_{maint}	Maintenance time per affected script	20 min
R	Execution cycles (regression runs)	12
Learning curve (extension)		
L	Learning rate — time multiplier per doubling	85% / 95%
b	Learning exponent, $b = -\log_2 L$	derived
n_0	Experience offset — units already completed	prior
Effort modifiers (extension)		
σ	Seniority factor — skill of who performs it	0.7–1.4
ρ	Client-process factor — governance overhead	1.0–1.8
κ	Tool-proficiency factor (AI and automation)	0.7–1.5
Cost layer (optional)		
w	Blended loaded labour rate (cost per effort-hour)	$/h
C_{fix}	One-time tooling cost (licence, onboarding)	$
C_{cyc}	Per-cycle tooling cost (amortised licence, compute)	$/cycle
Derived quantities and outputs		
e	Residual effort ratio per AI case	derived
B, B_{cost}	Break-even cycles (effort; cost-aware)	derived
$T_{\text{manual}}, T_{\text{AI}}$	Creation effort — manual; AI plus review	eff-hrs
$H_{\text{manual}}, H_{\text{auto}}$	Execution effort — manual; automated	eff-hrs
$S_{\text{creation}}, S_{\text{program}}$	Fraction of effort saved (creation; program)	output

These defaults are placeholders for explanation only. The savings the model reports depend entirely on these inputs, so substitute your own measured values for every one before drawing any conclusion. Different inputs give different results, by design.

1. Core Philosophy

Most efficiency claims answer one question: *how much faster is the new tool?* TAME asks a more precise one: *how many effort-hours does the new process remove, and how does that change as the work repeats?*

This distinction matters because creating a test case is paid once, while running it is paid every regression cycle. A model that blends the two hides where the savings actually come from. TAME is built on the principle:

> **Creation is paid once. Execution is paid every cycle.**

The Model in One Line

Before the details, here is the entire model in a single expression. Every time TAME uses — authoring, review, scripting, execution, triage — is a special case of one master form that holds all the variables at once:

$$\tau_x(i) \;=\; \sigma_x\,\rho_x\,\kappa_x \cdot t_x \cdot i^{-b_x}$$

Reading it left to right: σ, ρ and κ are the seniority, client-process and tool-proficiency modifiers; t_x is the intrinsic time the task takes in the neutral case; and i^{-b_x} is the learning discount that accrues with repetition. The document starts from the base — the neutral case where every modifier equals 1 and learning is switched off, so the master form is simply t_x — and then builds each term back in: the base times in §§2–8, the learning term in §9, and the context modifiers in §10, reassembled in full in §11. Leading with the complete form is deliberate: it shows that nothing is omitted, and that the simpler model which follows is this same equation with its extension terms switched off.

2. Manual Creation Cost

Formula

$$T_{\text{manual}} = N\,t_{\text{author}} \tag{1}$$

where

T_{manual} creation effort if every case is authored by hand
N number of test cases needed
t_{author} manual authoring time per case

Meaning

The baseline effort to author every test case by hand: the number of cases times the time to write one.

Example

$T_{\text{manual}} = 400 \times 20 = 8{,}000$ minutes = **133.3 effort-hours**.

Assumption

Authoring time per case is treated as an average. In practice it varies by complexity; sampling a representative batch is the way to fix it.

3. AI-Assisted Creation Cost

Formula

$$T_{\text{AI}} = k\,t_{\text{setup}} + (1-m)\,N\,(t_{\text{review}} + p\,t_{\text{correct}}) + m\,N\,t_{\text{author}} \tag{2}$$

where

T_{AI} creation effort with AI generation plus human review

k number of scenarios prompted into the AI tool
t_{setup} one-time setup per scenario — prompting, generation, data, vetting
m AI miss rate — fraction of needed cases the AI fails to produce
$t_{\text{review}}, p, t_{\text{correct}}$ review time, rework fraction, correction time

Meaning

AI does not remove the work; it shifts authoring to review. The tool drafts cases that a human reviews; a fraction need rework; and the cases the AI fails to produce fall back to full manual authoring.

Example

Setup $k\,t_{\text{setup}} = 200$; reviewed AI cases $0.90 \times 400 \times (4 + 0.3{\times}6) = 2{,}088$; missed cases $0.10 \times 400 \times 20 = 800$. Total $T_{\text{AI}} = 3{,}088$ min = **51.5 effort-hours**.

What t_{setup} really contains

The example treats setup as prompt-and-login time alone, but standing up AI generation for a scenario is more than prompting. In full it is the sum of four one-time costs:

$$t_{\text{setup}} = t_{\text{prompt}} + t_{\text{gen}} + t_{\text{data}} + t_{\text{vet}}$$

— login and prompting, AI generation plus the orchestration a human shepherds, provisioning the test data at least once, and vetting the generated cases before they are trusted. Counted in full, t_{setup} can approach the manual authoring time for a scenario. That does not by itself erase the saving, because it is paid once per scenario and spread over every case that scenario generates. AI creation pays off only once enough cases amortise the setup:

$$\frac{k\,t_{\text{setup}}}{N} \;<\; t_{\text{author}} - (t_{\text{review}} + p\,t_{\text{correct}})$$

The right side is the per-case advantage of reviewing over authoring; the left is the setup spread across cases. The deeper payoff is reuse on two levels: the setup amortises over cases in creation, and the resulting automated tests amortise again over cycles in execution (§12). The heavier the setup, the more the value depends on running the tests many times rather than on creating them once. Data provisioning needs one caution: provisioned once it is a setup cost, but when test data is consumed between runs, provisioning recurs and that recurring share belongs in the per-cycle execution cost (§12), not the one-time setup.

Why this was chosen

Review and rework are the real residual costs of AI generation. Modelling them explicitly keeps the saving honest rather than assuming the AI output is used as-is.

Alternatives considered

Pure generation, no review — rejected; unreviewed AI cases are not safe to run and overstate the saving. *Flat "X% faster" factor* — rejected; it hides the dependence on miss rate and rework, the variables that actually move the result.

4. Creation Savings and the Accuracy Ceiling

Formula

$$S_{\text{creation}} = \frac{T_{\text{manual}} - T_{\text{AI}}}{T_{\text{manual}}} = 1 - \frac{T_{\text{AI}}}{T_{\text{manual}}} \tag{3}$$

$$e = \frac{t_{\text{review}} + p\,t_{\text{correct}}}{t_{\text{author}}} \tag{4}$$

$$S_{\text{creation}} \approx (1-m)(1-e) \tag{5}$$

where

e residual effort ratio — share of manual effort an AI case still costs

From definition to closed form

Equation (3) is the definition of savings; (5) is what it becomes once the costs are substituted in. Writing the cost ratio and dividing each term of the numerator by $N\,t_{\text{author}}$:

$$\frac{T_{\text{AI}}}{T_{\text{manual}}} = \frac{k\,t_{\text{setup}}}{N\,t_{\text{author}}} + (1-m)\,\frac{t_{\text{review}} + p\,t_{\text{correct}}}{t_{\text{author}}} + m \;\approx\; (1-m)\,e + m$$

The middle ratio is exactly e from (4); the first term — the one-time setup spread across all N cases — is about 2.5% at the defaults, small enough to drop. Substituting back and factoring out $(1-m)$ gives (5): the saving is coverage $(1-m)$ times per-case efficiency $(1-e)$ — the product that makes the accuracy ceiling visible.

Key property

> **Savings are bounded by accuracy, not speed. If the AI misses 20% of cases, the saving cannot exceed 80% no matter how fast review becomes.**

Example

With $e = (4 + 0.3{\times}6)/20 = 0.29$ and $m = 0.10$: $S_{\text{creation}} \approx (1-0.10)(1-0.29) \approx \mathbf{61\%}$ — saving 81.9 of 133.3 effort-hours.

Setup time and what to automate first

The setup term does not touch the execution break-even — that is purely an execution quantity. What it decides is whether the creation stage pays at all, a per-scenario question: a scenario's one-time setup must be repaid by the cases it yields. Per scenario the saving is:

$$S_{\text{scenario}}(c) = (1-m)(1-e) - \frac{t_{\text{setup}}}{c\,t_{\text{author}}}$$

The first term is the setup-free ceiling — about 64% at the defaults; the second is the setup spread over the c cases that share it. Setup breaks even when the two are equal:

$$c^{\star} = \frac{t_{\text{setup}}}{(1-m)(1-e)\,t_{\text{author}}}$$

At the defaults the denominator is about 12.8 minutes, so $c^{\star}$ is roughly 0.8, 1.6 and 2.4 cases for a t_{setup} of 10, 20 and 30 minutes. Below that the scenario loses; above it the saving climbs toward the ceiling.

Per-scenario creation saving, by cases yielded (rows) and setup time (columns)

Cases c	10 min	20 min	30 min
1	+14%	−36%	−86%
2	+39%	+14%	−11%
5	+54%	+44%	+34%
10	+59%	+54%	+49%
20	+61%	+59%	+56%
50	+63%	+62%	+61%

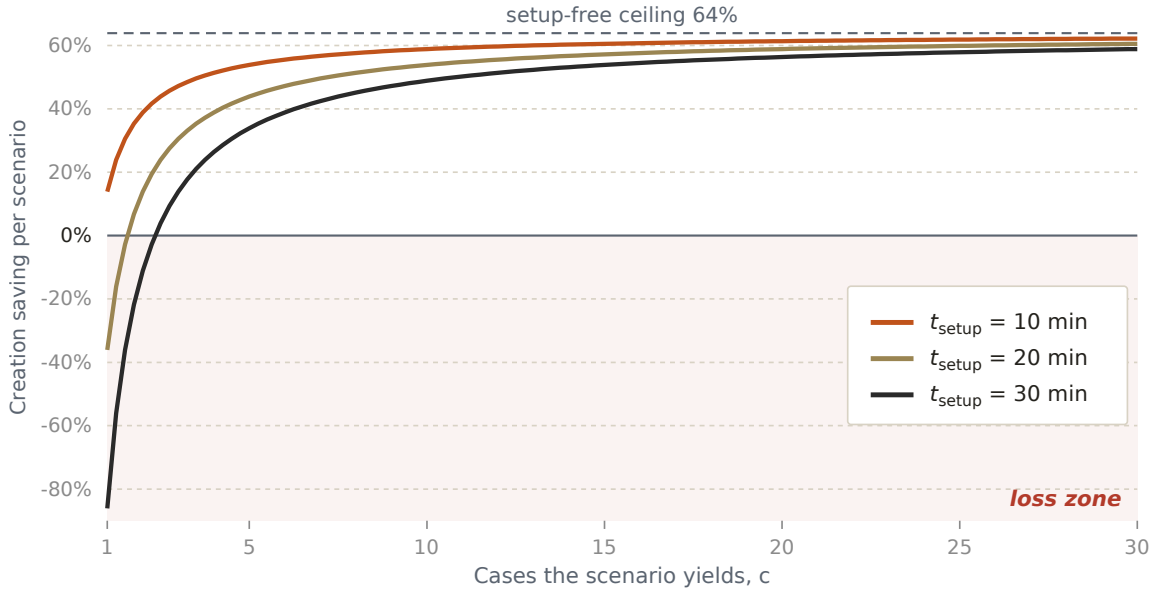

Figure 1. Creation saving per scenario against the number of cases a scenario yields, for three setup times. A heavy setup (30 min) sits in the loss zone until the scenario yields about two to three cases; a light one (10 min) turns positive almost at once. All three climb toward the same 64% setup-free ceiling.

> **Automate the low-hanging scenarios first: low setup, many cases. They pay back immediately and fund the harder ones later.**

This is why the order of automation matters as much as the decision to automate. Front-load the scenarios that yield early gains; defer the complex, low-yield ones until the suite — and the team's setup speed — has matured (§9). Chasing the hardest scenarios first is the most common way a promising automation effort posts a loss in its opening project.

5. Manual Execution Cost

Formula

$$H_{\text{manual}} = R\,N\,t_{\text{exec}} \tag{6}$$

where

H_{manual} execution effort if every case is run by hand each cycle
R number of execution cycles (regression runs)

Meaning

The recurring baseline: a human re-runs every case, every regression cycle. This cost is paid R times, which is what makes execution the dominant term over a project.

Example

$H_{\text{manual}} = 12 \times 400 \times 12 = 57{,}600$ minutes = **960 effort-hours**.

Assumption

Every case is executed every cycle. Suites that run only a subset per cycle can scale R per case, or use an effective average.

6. Automated Execution Cost

Formula

$$H_{\text{auto}} = a\,N\,t_{\text{script}} + R\big[\,a\,N\,(t_{\text{triage}} + \mu\,t_{\text{maint}}) + (1-a)\,N\,t_{\text{exec}}\,\big] \tag{7}$$

where

H_{auto} execution effort with automation
a automation coverage — share of cases scripted
t_{script} scripting time per automated case (one-time)

Meaning

Automation front-loads a large scripting cost, then runs for almost nothing each cycle. The unattended machine run itself costs zero effort-hours; the only recurring human costs are triaging results, repairing broken scripts, and executing the cases that were never automated.

Example

Upfront scripting $0.70 \times 400 \times 60 = 16{,}800$; per cycle: automated upkeep $280 \times 2.6 = 728$ plus manual remainder $0.30 \times 400 \times 12 = 1{,}440$, so 2,168 per cycle. $H_{\text{auto}} = 16{,}800 + 12 \times 2{,}168 = 42{,}816$ min = **713.6 effort-hours** (of which 280 h is one-time scripting).

Why this was chosen

Separating one-time scripting from per-cycle upkeep is what reveals the payback dynamic. Folding them together would hide why automation loses at low cycle counts and wins at high ones.

Alternatives considered

Zero per-cycle cost — rejected; ignores triage and maintenance, the costs that quietly erode automation ROI. *Maintenance as a one-time cost* — rejected; scripts break as the application changes, so upkeep recurs each cycle.

7. Break-Even Cycles

Formula

$$B = \frac{t_{\text{script}}}{t_{\text{exec}} - t_{\text{triage}} - \mu\, t_{\text{maint}}} \tag{8}$$

Meaning

The number of cycles needed for one automated script to repay its scripting cost. The manual cases and the coverage fraction appear identically on both sides of the comparison and cancel out. Break-even is fundamentally an amortisation question: how many cycles must share the one-time cost before the amortised figure drops below the recurring manual expense.

Key property

> **Break-even is independent of test volume and automation coverage.**

Example

$B = 60/(12 - 1 - 1.6) =$ **6.4 cycles**. Past this point the suite has repaid its scripting cost and every further cycle widens the gap against manual.

Failure case

If $t_{\text{exec}} - t_{\text{triage}} - \mu\, t_{\text{maint}} \leq 0$, automated upkeep costs as much as a manual run and automation never pays back. The model flags this explicitly.

8. Combined Program Savings

Formula

$$H_{\text{old}} = N\, t_{\text{author}} + R\, N\, t_{\text{exec}} \tag{9}$$

$$H_{\text{new}} = T_{\text{AI}} + H_{\text{auto}} \tag{10}$$

$$S_{\text{program}} = \frac{H_{\text{old}} - H_{\text{new}}}{H_{\text{old}}} \tag{11}$$

where

H_{old} total effort if everything stays manual
H_{new} total effort with AI plus automation
S_{program} fraction of total effort saved over the program

Worked example

Stage	Manual (h)	New (h)	Saved (h)	Saved
Creation (one-time)	133.3	51.5	81.9	61%
Execution (12 cycles)	960.0	713.6	246.4	26%
Combined program	**1,093.3**	**765.1**	**328.3**	**30%**

Execution shows only 26% at 12 cycles because upfront scripting and the manual remainder hold it down. Raise the cycle count or the coverage and the figure climbs steeply — the single largest lever in the model is R.

Combined program saving across a range of cycle counts

Cycles R	Manual (h)	New (h)	Combined saved
1	213	368	−72%
4	453	476	−5%
6	613	548	+11%
12	**1,093**	**765**	**30%**
24	2,053	1,199	42%
48	3,973	2,066	48%
100	8,133	3,945	52%

The rows are not multiples of the first because each total splits into a one-time cost plus a per-cycle cost: Manual $= 133.3 + 80R$ and New $= 331.5 + 36.1R$ hours. Only the per-cycle term scales. The one-time creation (133.3 h manual; 51.5 h of AI creation plus 280 h of upfront scripting for the new process) is paid once, not once per cycle.

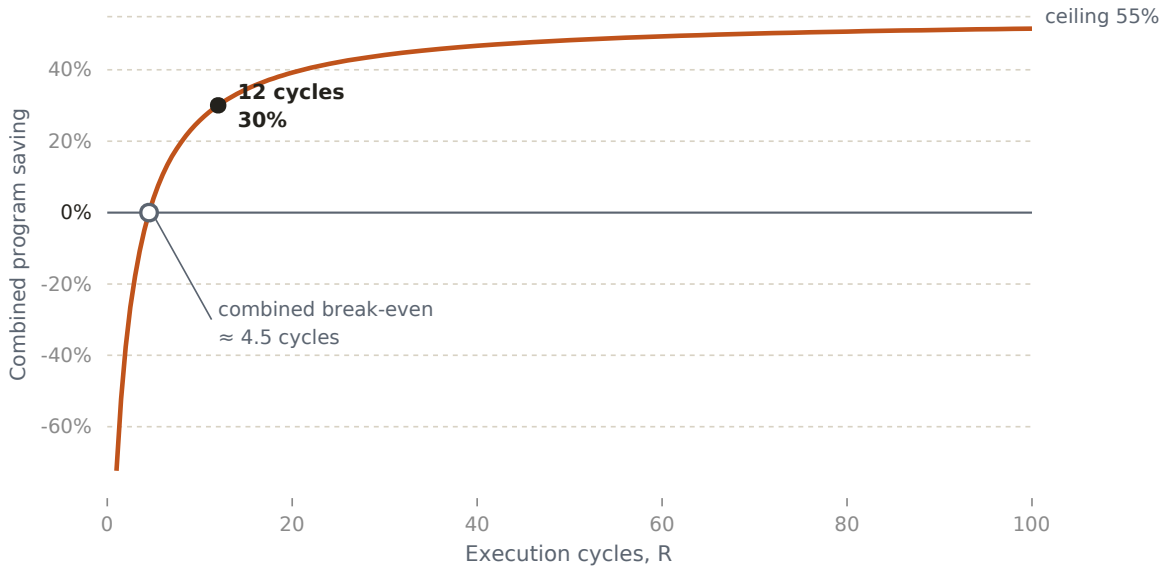

Figure 2. Combined program saving against execution cycles. The curve is negative below about four to five cycles — automating the whole program costs more than staying manual until then — then climbs steeply and flattens toward a ceiling near 55%. This combined break-even (≈ 4.5 cycles) sits below the execution break-even $B = 6.4$ of (8), because the one-time creation saving gives the program a head start.

The steep early rise is the point: most of the value is won in the first ten to twenty cycles, which is why R — not coverage or per-case efficiency — is the lever that moves the result most.

9. Learning Curve Extension *(optional)*

The base model holds every per-unit time constant. In reality, constructive work speeds up with repetition — most sharply for automation scripting, where the first scripts build reusable scaffolding that later scripts inherit. TAME models this with a power-law learning curve that switches off cleanly, reducing exactly to the base model.

Formula

$$t_i = t_1\, i^{-b}, \qquad b = -\log_2 L \tag{12}$$

$$T_{\text{learn}}(Q) \approx t\, n_0^{\,b}\, \frac{(n_0+Q)^{1-b} - n_0^{1-b}}{1-b} \tag{13}$$

where

t_i time to perform the i-th repetition of a learnable activity
Q number of new repetitions in this project
n_0 experience offset — repetitions already completed
t current per-unit time at the team's present skill

Meaning

Each repetition of a learnable activity is a little faster than the last, following a constant percentage improvement per doubling of cumulative experience. The offset n_0 says how far up that curve the team already sits — so the entered rate keeps its natural meaning: what the task costs us now.

Parameters

L — learning rate, the time multiplier per doubling; default 85% for scripting, 95% for authoring and review. n_0 — experience offset; small = cold start (new framework), large = seasoned team ($\approx$ flat). $L = 100\%$ ($b = 0$) reduces to $T_{\text{learn}} = t \times Q$: the base model exactly. Learning is off by default.

Where it applies

Scripting — primary; strongest and best-documented learning effect. *Authoring and review* — optional, mild (high L), applied to both the manual baseline and the AI path so savings are not inflated. *Execution, triage, maintenance* — flat; mechanical and bounded.

Example

Scripting at $t = 60$ min, $L = 85\%$ (so $b = 0.234$), $n_0 = 40$, over $Q = 280$ scripts gives $T_{\text{learn}} \approx 12{,}260$ min $\approx$ **204 effort-hours**, against 280 h at a flat rate. The combined program saving rises from 30% to $\approx$ 37%.

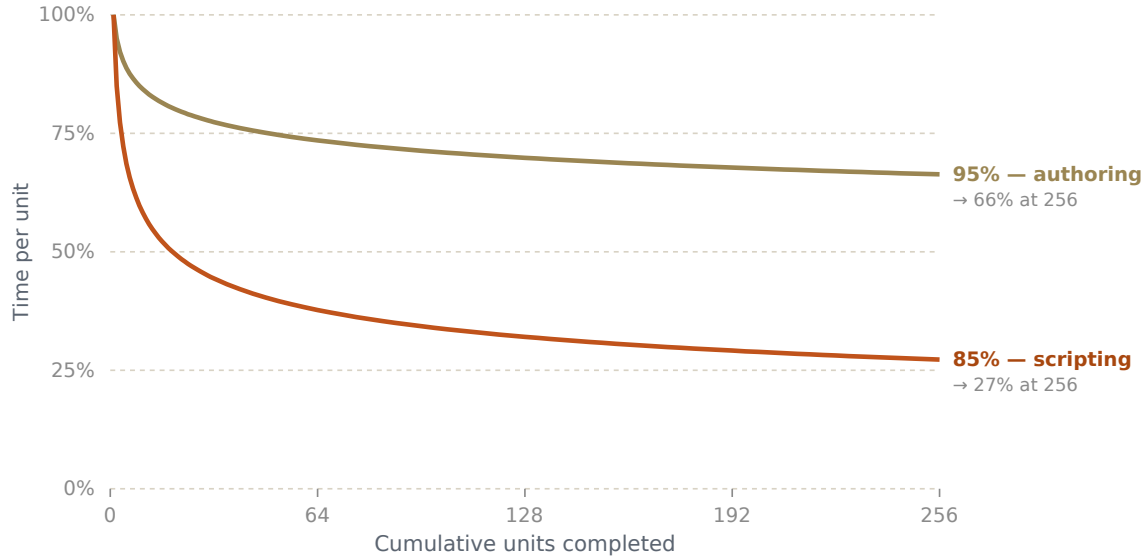

Figure 3. Power-law learning. Scripting (85%) compounds through reuse, falling to about 27% of first-unit time over eight doublings; authoring and review (95%) have little reusable artifact, so per-unit time bends only to about 66%. The gap — scripting steeper than authoring — is the substantive claim, not the exact percentages.

Why a power law was chosen

Repeated constructive work has followed a power law since Wright (1936): a straight line on log–log axes, a constant percentage improvement per doubling of experience. This matches how a scripting framework actually matures — early effort builds shared structure, later effort reuses it.

> **Learning rewards repetition. Scripting repeats the most, so it learns the most.**

10. Effort Modifiers: Seniority, Client Process, Tool Proficiency

The base model treats each per-unit time as a fixed average. Three real-world factors shift those times systematically: who does the work (seniority), where it is done (the client's process), and how well the team knows the tools. TAME folds them in as dimensionless multipliers on the base times, each defaulting to 1 so the base model is the neutral case.

Formula

$$t_x^{\text{eff}} = \sigma_x \, \rho_x \, \kappa_x \, t_x \tag{14}$$

where

t_x^{eff} effective time for activity x after modifiers
σ_x seniority factor (skill of who performs it)
ρ_x client-process factor (governance overhead)
κ_x tool-proficiency factor

Every per-unit time t in §§2–8 is read as its effective value t^{eff}. The factors apply per activity; an activity a factor does not touch simply takes 1.

Where each factor applies

Activity	σ seniority	ρ process	κ tools
Manual authoring	✓	✓	—
AI review	✓	○	κ_{AI}
Rework	✓	—	κ_{AI}
Prompt / setup	○	—	κ_{AI}
Manual execution	○	✓✓	—
Scripting	✓	—	κ_{auto}
Triage	✓	—	κ_{auto}
Maintenance	✓	—	κ_{auto}
Automated run	—	○	—

✓✓ strong ✓ applies ○ minor — none

Seniority (σ)

The skill level of whoever performs the activity. $\sigma < 1$ is faster (senior), $\sigma > 1$ is slower (junior). The effect is strongest on judgement-heavy work — review, rework, scripting — and weakest on mechanical execution. Typical range 0.7–1.4. σ scales effort-hours only; to convert to cost, weight each activity's effective hours by the loaded rate of the role that performs it.

Client process (ρ)

The client's governance overhead per unit of work: environment access, approvals, evidence capture, traceability, sign-offs. $\rho = 1$ is a lightweight or agile client; ρ rises toward ∼1.8 in heavily regulated programs. The asymmetry is the important part: a human pays ρ on every manual run, while an automated run emits its logs and audit trail automatically, so ρ barely touches it.

Governance is paid on every manual run, but essentially once by automation.

So high process maturity multiplies the automation advantage — central for regulated programs such as SAP finance, where every manual execution carries mandatory documentation. Concretely, ρ rises with the compliance regime; common ones include:

- **SOX §404** — IT general controls on financial-reporting systems such as SAP FICO; every change is documented, tested, and signed off.
- **FDA 21 CFR Part 11 with GAMP 5** — computerized-system validation in life sciences (IQ/OQ/PQ protocols, full traceability), the heaviest common regime.
- **PCI-DSS** — controlled testing and evidence for payment-card systems.
- **GDPR / HIPAA** — data handling, masking, and privacy controls in test environments.
- **SOC 1 / SOC 2 and SR 11-7** — audit and model-risk controls in banking and financial services.
- **IEC 62304 / DO-178C / ISO 26262** — verification rigour in medical, avionics, and automotive software.

Tool proficiency (κ)

How well the team knows the specific tools — the AI generator (κ_{AI}) and the automation framework (κ_{auto}). It works through two channels. **Time:** κ multiplies tool-mediated times. **Quality:** a fluent team prompts better and writes sturdier scripts, lowering the rework, miss, and maintenance fractions:

$$p_{\text{eff}} = \kappa_{\text{AI}}\, p, \qquad m_{\text{eff}} = \kappa_{\text{AI}}\, m, \qquad \mu_{\text{eff}} = \kappa_{\text{auto}}\, \mu \tag{15}$$

κ is today's static proficiency; the learning curve (§9) is its trajectory. For scripting, set κ_{auto} for the current rate and n_0 for how fast it improves — do not count the same gain in both.

Why multiplicative

The factors compound rather than add: a senior expert at a low-governance client is fast on every count, and the effects stack proportionally to task size. Multiplicative factors keep each cause separable and auditable, and collapse to the base model when set to 1.

Example

At a heavily governed client, $\rho \approx 1.6$ on manual execution lifts it to $12 \times 400 \times 12 \times 1.6 \approx$ **1,536 effort-hours**, up from 960. Automated upkeep is almost unchanged, so the program saving rises from 30% to roughly 44% and break-even arrives sooner.

11. The Complete Model: Base Plus Extensions

Having developed each term on its own — the base times (§§2–8), the learning curve (§9), and the effort modifiers (§10) — this section reassembles them into the complete form previewed in §1. The base is the starting point, and the learning and context terms are extensions switched on top of it.

The master time

Every time that appears anywhere in TAME is, in full generality, one expression combining the intrinsic cost, the context modifiers of §10, and the learning term of §9:

$$\tau_x(i) = \sigma_x\, \rho_x\, \kappa_x \cdot t_x \cdot i^{-b_x} \tag{16}$$

where

$\tau_x(i)$ effective time for the i-th unit of activity x
t_x intrinsic time for activity x — the neutral, first-unit cost
b_x learning exponent for activity x, $b_x = -\log_2 L_x$

Three independent influences: **intrinsic cost** (the neutral first-unit time), **context** (who does it, where, how skilled — each 1 in the neutral case), and **experience** (the learning discount; $b_x = 0$ for activities that do not learn).

From per-unit time to the totals

The master time is per repetition. The stage totals need the average per-unit time across a whole batch — the modifiers times the learning-integrated base:

$$\bar{\tau}_x = \frac{1}{n}\sum_i \tau_x(i) = \sigma_x\, \rho_x\, \kappa_x \cdot \frac{1}{n}\sum_i t_x\, i^{-b_x} \tag{17}$$

The inner sum is the cumulative learning form of (13); with learning off it equals the count times the intrinsic time, so the batch-effective time reduces to the modifiers times t_x.

The master totals

$$T_{\text{create}} = k\,\tau_{\text{setup}} + (1-m)\,N\,(\tau_{\text{review}} + p\,\tau_{\text{correct}}) + m\,N\,\tau_{\text{author}} \tag{18}$$

$$H_{\text{exec}} = a\,N\,\tau_{\text{script}} + R\big[\,a\,N\,(\tau_{\text{triage}} + \mu\,\tau_{\text{maint}}) + (1-a)\,N\,\tau_{\text{exec}}\,\big] \tag{19}$$

These are (2) and (7) with every constant time replaced by its batch-effective value. The quality terms travel the same way — p, m, μ carry the tool-proficiency channel of (15).

Consistency: the neutral case recovers the base

Setting $\sigma_x = \rho_x = \kappa_x = 1$ and $b_x = 0$, the master time collapses to the intrinsic time:

$$\tau_x(i) = (1)(1)(1)\cdot t_x \cdot i^0 = t_x \tag{20}$$

and the complete totals (18) and (19) return exactly to the constant-rate equations (2) and (7). The base is the complete model with its extensions switched off — precisely as the one-line preview promised.

One formula, different inputs → different savings

Setting	Inputs changed from neutral	Saving
Neutral baseline	all $\sigma, \rho, \kappa = 1$; learning off	30%
+ scripting learning	$L = 85\%$, $n_0 = 40$	≈ 37%
+ heavy governance	$\rho \approx 1.6$ on manual execution	≈ 44%
+ **senior, tool-fluent team**	$\sigma \approx 0.85$, $\kappa \approx 0.8$	**higher still**

The base model is one configuration. Different inputs give different savings.

Every row uses the identical formula; only the inputs differ. The intended use is for a reader to set the variables to their own situation and read off their own number. The model's job is not to assert a single percentage — it is to be honest about which variables move the result and to let each reader arrive at theirs.

12. Tooling Cost and the Investment Decision

Effort-hours are the headline, but the go/no-go for automation also turns on money — the loaded cost of the hours saved, and the tool's own licence and infrastructure cost. This section adds the cost layer and turns the effort break-even of §7 into a money break-even management can act on.

From effort-hours to money

$$\text{Cost} = w\,H + C_{\text{fix}} + C_{\text{cyc}}\,R \tag{21}$$

where

w blended loaded labour rate (cost per effort-hour)

H effort-hours from the model (manual or new process)
$C_{\text{fix}}, C_{\text{cyc}}$ one-time and per-cycle tooling cost

One-time C_{fix}: platform licence onboarding, implementation and CI integration, initial environment build, one-off training. Per-cycle C_{cyc}: subscription and per-seat licences amortised to a cycle, runner/compute or cloud-grid minutes, and any per-execution vendor charge.

Cost-aware break-even

$$B_{\text{cost}} = \frac{w\,a\,N\,t_{\text{script}} + C_{\text{fix}}}{w\,a\,N\,(t_{\text{exec}} - t_{\text{triage}} - \mu\,t_{\text{maint}}) - C_{\text{cyc}}} \tag{22}$$

The numerator is the total upfront automation investment; the denominator is the per-cycle cost saving net of tooling. With both tooling terms zero, dividing through recovers the effort break-even (8) exactly.

Licence cost raises the break-even two ways: a fixed cost lifts the numerator; a per-cycle cost shrinks the denominator.

The viability test and net value

$$V(R) = \left[\,w\,a\,N\,(t_{\text{exec}} - t_{\text{triage}} - \mu\,t_{\text{maint}}) - C_{\text{cyc}}\,\right] R - \left(w\,a\,N\,t_{\text{script}} + C_{\text{fix}}\right) \tag{23}$$

V is positive precisely when the planned cycles exceed the cost-aware break-even. First the per-cycle saving must be positive; second the program must run more cycles than B_{cost}.

Worked example

With w = \$80/h, C_{fix} = \$20,000, C_{cyc} = \$500: upfront investment \$42,400; per-cycle saving net of tool \$3,009; B_{cost} = 42,400/3,009 = **14.1 cycles**. The effort break-even was 6.4 — licence and compute more than double it. At a planned 12 cycles, automation would lose roughly $V \approx -\$6{,}300$; it becomes worth it only beyond ∼14 cycles.

Amortisation: how much to invest

Manual execution is a raw operating expense — the same cost every cycle, capitalising into nothing. Automation scripting is a capital-like outlay — spent once, then spread across every cycle it serves.

$$A_{\text{manual}} = w\,N\,t_{\text{exec}} \tag{24}$$

$$A_{\text{auto}}(R) = \frac{w\,a\,N\,t_{\text{script}} + C_{\text{fix}}}{R} + g \tag{25}$$

$$g = w\left[\,a\,N\,(t_{\text{triage}} + \mu\,t_{\text{maint}}) + (1-a)\,N\,t_{\text{exec}}\,\right] + C_{\text{cyc}}$$

The automation cost per cycle is a hyperbola in R: the upfront term shrinks as more cycles share it, so the cost per cycle falls toward the running-cost floor g, while the manual line stays flat. The two cross exactly at the cost-aware break-even:

$$A_{\text{auto}}(R) < A_{\text{manual}} \quad\Longleftrightarrow\quad R > B_{\text{cost}} \tag{26}$$

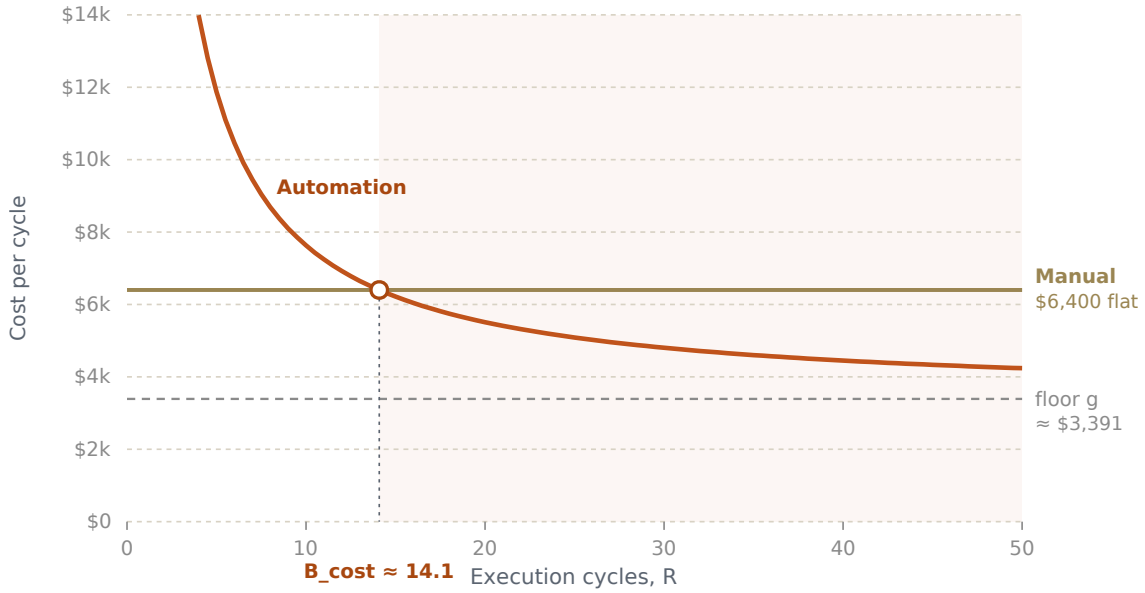

Figure 4. Cost per cycle: a flat manual expense against the amortised automation hyperbola, falling toward the running-cost floor g and crossing the manual line at the cost-aware break-even, ≈ 14.1 cycles in the worked example. Left of the crossing, automation is the more expensive choice.

> **Manual execution is expensed every cycle; automation is a one-time asset amortised across them.**

How many cycles does a project actually run?

$$R = R_{\text{impl}} + R_{\text{reg}}, \qquad R_{\text{reg}} = f \cdot W \tag{27}$$

where

R_{impl} implementation cycles — the SIT and UAT passes during the project
R_{reg} ongoing regression cycles after go-live
f, W regression runs per period; number of periods the suite is maintained

A typical implementation runs only a handful of passes — two system-integration cycles (SIT1, SIT2), a user-acceptance cycle (UAT), and often a pre-go-live dry run — so R_{impl} is about three to four. Everything beyond is regression. Since the cost-aware break-even was ∼14 cycles but implementation supplies only three or four, the investment is not recovered by the project that builds the suite — it is recovered, if at all, by the regression that follows. Automation pays back only when:

$$R_{\text{impl}} + R_{\text{reg}} \geq B_{\text{cost}} \iff R_{\text{reg}} \geq B_{\text{cost}} - R_{\text{impl}} \tag{28}$$

> **Automation is rarely repaid by the project that builds it — it is repaid by the regression cycles that follow.**

Guiding the decision

Estimate the loaded rate w for the people who run and maintain the suite. Get the tool economics from the vendor quote, split into one-time C_{fix} and recurring C_{cyc}. Compute the

per-cycle saving net of tooling — if zero or negative, stop. Otherwise compute B_{cost} and compare to the cycles you genuinely expect — implementation passes plus realistic regression, not the horizon you hope for. Invest only if expected cycles comfortably exceed the break-even, leaving margin for maintenance spikes and coverage that falls short of plan.

13. System Behavior

Few cycles (R below B). Scripting cost not yet repaid; automation shows a net loss; stay manual. **Many cycles (R well above B).** Upfront scripting amortizes away; each cycle is near-free; percent saved approaches the coverage limit. **Low coverage (small a).** Most cases still run manually each cycle; saving is capped well below 100% regardless of cycles.

14. Calibration

The defaults are industry-typical priors, not measured truth. Before reporting, sample real cases to fix the parameters that matter most:

- t_{author} and t_{review} — time a small batch by hand and by review to set the creation ratio.
- m — the AI miss rate is the hard ceiling on creation savings and the figure most often forgotten.
- t_{script} and μ — scripting effort and flaky-test maintenance are where optimistic automation cases break down.
- a and R — be honest about how many cases can truly be automated and how often the suite runs.
- L and n_0 — if learning is enabled, fit them from a script-time log on log–log axes; the slope gives b (hence L), and how far in you already are gives n_0.
- σ, ρ, κ — set seniority from the staffing mix, the client-process factor from the documentation each run carries, and tool proficiency from the team's familiarity with the tools.

15. Scope and Limitations

- **Effort-hours only.** Faster wall-clock turnaround from unattended runs is a separate throughput benefit, intentionally excluded.
- **Linear per-cycle costs.** Maintenance is modelled as a steady fraction; it can spike around major releases.
- **Learning is optional and parametric.** When enabled it assumes a single stable learning rate per activity; abrupt tooling changes or heavy turnover are only approximated.
- **Quality effects excluded.** Earlier defect detection and broader coverage have real value but sit outside this effort-based model — treat them as unquantified upside.
- **Inputs are estimates.** The output is only as good as the calibration in §14; present it as a defensible estimate with stated assumptions, not a measurement.

16. Conclusion

TAME shifts the question from an informal speed claim to a measured one: *how many effort-hours does the new process remove, stage by stage, as the work repeats?* The model combines a one-time, accuracy-bounded creation saving; a recurring execution saving that compounds with cycles; and a volume-independent break-even point. It rests on a simple philosophy:

> **Manual testing pays by the cycle. Automation pays once, then runs free.**

Rationale

This appendix consolidates the reasoning behind every modelling choice. Each entry states the decision, the reasoning, and the main alternative rejected.

Two stages

Separate creation from execution: their cost structures are categorically different — creation incurred once, execution every cycle — so a single blended figure hides where the savings come from. Rejected: a one-number "X% faster."

Effort-hours as the unit

Measure human effort, not calendar time or money. The mandate is effort saved; wall-clock turnaround would inflate the figure if mixed in, and money depends on rates that vary by organisation. Effort-hours convert cleanly to cost later by weighting with loaded rates (§12).

Linear in case count and cycles — and why not exponential

Authoring or running one case does not change the cost of the next: no shared computation, no feedback loop, so no compounding mechanism justifies exponential growth. An exponential form here would be curve-fitting without a cause. Linearity is also what makes break-even independent of volume and coverage. The only legitimate departure is the mild sub-linear discount from learning, handled separately (§9).

Rework as an expected value

Rework enters as the expected cost per case. TAME predicts an aggregate over many cases, and by the law of large numbers the expected value is an accurate predictor of a sum — the same logic an insurer uses pricing on expected loss.

A missed case costs full authoring time

The conservative choice — a reviewer with context might author a missed case slightly faster, so assuming full cost cannot overstate the saving. Adjust downward only with evidence.

The capped form $(1 - m)(1 - e)$

Express the creation saving as two bounded factors — coverage and per-case efficiency. This makes explicit that accuracy, not speed, sets the ceiling. A raw speed-up hides the miss-rate ceiling and was rejected.

Why a power law for learning — the alternatives, formally

A learning law must stay positive, reproduce constant-improvement-per-doubling, and be parsimonious. One diagnostic separates the candidates — the ratio of times one doubling apart.

For a power law it is constant:

$$\frac{t(2i)}{t(i)} = \frac{(2i)^{-b}}{i^{-b}} = 2^{-b} = L \quad \text{(constant)}$$

a straight line on log–log axes (Wright, 1936). *Exponential to a floor* measures improvement against the floor, so the ratio decays with experience and the curve flattens too early — understating long-run gains; kept only as the documented variant when a hard floor exists. *Logarithmic* is unbounded below — it crosses zero and turns negative, physically impossible; rejected outright. *Logistic S-curve* needs two hard-to-identify parameters and models slow early improvement, the opposite of the steep initial drop real learning shows; the offset n_0 already absorbs any head start. Only the power law stays positive, keeps the per-doubling ratio constant, and does so with one parameter.

Anchoring learning on current skill

Anchor on the team's present rate through an experience offset, not a first-unit time. A "first script ever" time is nearly impossible to estimate and produces absurd deflation across hundreds of units; anchoring on what a script costs now keeps the entered rate meaningful and lets the offset carry how far up the curve the team sits.

Why 85% for scripting and 95% for authoring and review

Which rate fits an activity is governed by one question: how much does performing it once create reusable leverage for the next time? Scripting is constructive with high leverage — the first scripts build the framework (page objects, locator helpers, fixtures, CI wiring) that every later script inherits — exactly the mechanism behind the steep 80–85% curves long reported for construction and tooling. Authoring is cognitive with low leverage — each case targets different functionality, with a floor set by human comprehension speed that repetition cannot compress — pointing to a shallow curve near 95%. The gap, not the exact figures, is the substantive claim: over eight doublings an 85% curve cuts per-unit time to ∼27%, a 95% curve only to ∼66%. Both are priors to fit from a time log before production.

Effort modifiers multiply, not add

Each is a proportional effect on the same unit of work, and proportional effects compound — a senior 15% faster at a client 60% heavier lands at 0.85×1.6, not $0.85 + 0.6$. Multiplication keeps each cause separable and auditable and reduces to the base model at 1. Additive overheads fit only genuinely fixed steps, kept as a variant.

Tool proficiency scales the quality terms too

Fluency with a tool is not only faster but better — a skilled prompter elicits fewer misses and less rework, a skilled engineer writes sturdier scripts. Limiting κ to time would miss its largest effect. It is tied to the learning offset so the same improvement is not counted twice.

Why the client-process factor reaches about 1.8

$\rho = 1.8$ means roughly 0.8 hour of documentation, review and sign-off for every hour of testing — high-risk validated environments (GAMP Category 5, SOX-controlled finance) where every execution is pre-approved, evidenced step by step, traced to a requirement, four-eyes reviewed and archived. Reported compliance overhead runs ∼+30% for light regimes to +80–100% for the strictest. The asymmetry is what makes the factor matter: automation produces most of this evidence as a by-product of running, so a higher ρ widens the automation advantage rather than narrowing it.

The machine run counts as zero effort-hours

The model measures people's time, and a script on a server consumes none. The recurring human costs that remain — triage and maintenance — are modelled explicitly. If runs are supervised in practice, that time belongs in the triage term, not at zero.

Every factor defaults to 1

Every modifier, and the learning switch, defaults to the neutral value, so the base model is recoverable exactly and added realism is opt-in and auditable.

Prepared as a companion to test-automation-savings.xlsx. Figures recompute live in the workbook.