A Practitioner Framework for Automation Tool Adoption Across Web, Mobile, API, Mainframe, and Database Applications

Sridhar Gajavelli
Independent Researcher
Texas, USA

ABSTRACT
Test automation tool selection is usually reported as a feature checklist for one technology, most often web UI automation. That framing does not fit enterprise financial-services environments, where a single quality-assurance function must validate a mainframe policy administration system, a web-based new-business workflow, a native mobile application, an ACORD TX103 XML messaging layer, and the Oracle/DB2 databases underneath them. This paper presents a framework for evaluating and migrating automation from a licensed tool (UFT One) to a single codeless platform (accelQ) across these five layers. The framework has two parts: (i) a layer-by-layer evaluation that tests licensing cost, coverage, codeless authoring, and self-healing claims against each technology layer separately, and (ii) a five-phase, gate-controlled migration method that uses risk-based prioritization, per-layer pilots, parallel-run UFT-versus-accelQ outcome comparison, and staged license retirement, so that automated coverage never drops below its pre-migration level. The framework is evaluated on an industrial insurance platform using four research questions covering cost, coverage parity, authoring and maintenance effort, and adoption. Practitioners can reuse the method for any tool-to-tool automation migration.

General Terms
Software Testing, Test Automation, Software Quality Assurance, Measurement, Experimentation.

Keywords
Test automation migration; UFT; accelQ; codeless automation; self-healing locators; parallel-run validation; mainframe testing; ACORD TX103; insurance policy administration.

1. INTRODUCTION
In an insurance core-administration environment, the testing surface does not sit in one technology. Policy administration runs on a mainframe platform. New-business intake runs through a web-based workflow engine. Servicing increasingly happens through a native mobile application. The systems exchange XML messages built to the ACORD TX103 standard [3], and every path must ultimately be checked against the same relational database tables. A tool bought to automate one of these layers well does not automatically cover the rest.

In the organization studied here, UFT One [1] had covered much of this surface for years. The motivation to replace it was not technical failure but cost and operating model. UFT is licensed per seat, native mobile-device testing requires a separately licensed device-lab product, and test assets are maintained in VBScript. Open-source alternatives such as Selenium remove license cost but introduce a significant dependency-management burden, including separately versioned browser drivers per browser and release [10], [11], and cover only part of the five-layer surface. The organization also wanted a tool that insurance domain specialists, rather than scripting specialists, could adopt directly. These three drivers (license cost, open-source dependency overhead, and the need for codeless authoring) led to the evaluation of accelQ [2] as a single replacement platform.

Most published tool comparisons evaluate a single technology layer and stop at tool selection [12]. They seldom report how a production automation estate is moved from one tool to another without a coverage gap, or measure the outcome. This paper addresses both gaps. Its contributions are:

- A layer-decomposed evaluation method that tests a candidate tool separately against each of five technology layers (web, mobile, API, mainframe, database) instead of issuing one generic verdict.
- A five-phase, gate-controlled migration method (inventory, readiness, per-layer pilot, parallel-run validation, staged license retirement) that keeps automated coverage at or above its pre-migration level throughout.
- An empirical evaluation on an industrial insurance platform against four research questions on cost, coverage parity, effort, and adoption, reported with tabular and graphical results.

Section 2 reviews related work. Section 3 describes the environment. Sections 4 and 5 present the evaluation and migration methods. Section 6 defines the experimental setup. Section 7 reports results, Section 8 discusses them, Section 9 covers threats to validity, and Section 10 concludes.

2. RELATED WORK

2.1 Codeless and AI-Assisted Automation
Codeless automation platforms aim to let testers build tests through visual, model-based designers instead of program code. Gatla et al. [7] report that codeless approaches lower the skill barrier for test authoring in development QA, and Martins et al. [6] review how AI techniques are being applied across the testing life cycle. Self-healing object identification, in which locators adapt automatically when an element's attributes change, is proposed in [9] as a response to the maintenance cost of brittle UI scripts.

2.2 Tool Comparison and ROI
Comparative studies of testing tools [12] typically score tools on features, cost, and usability within one application type. Selenium WebDriver's architecture and framework design are described in [10], and García et al. [11] quantify the driver-management problem that motivates dedicated tooling. De Reanzi and Thangaiah [8] survey automation ROI and find that returns depend heavily on maintenance cost and organizational

adoption, not license price alone.

2.3 Tool Migration Practice

Practitioner guidance on proofs of concept [4] recommends agreed criteria, a minimum sample, and a written outcome report before a tool decision. Song [5] documents a Cypress-to-Playwright migration that piloted each test against its predecessor, ran both frameworks in CI during the transition, and retired the old framework only after coverage moved. None of these sources addresses migration across heterogeneous layers that include mainframe and message-level testing. That is the setting this paper studies.

3. TESTING ENVIRONMENT AND PROBLEM STATEMENT

The system under test is a life and annuity insurance platform. Figure 1 shows its five technology layers and the data flow between them. Table 1 summarizes what automation must validate in each layer and the main automation risk.

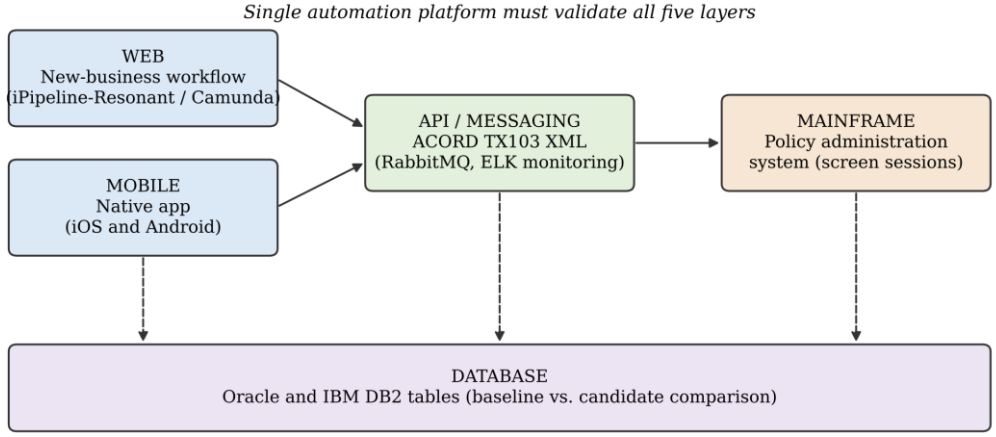


Fig 1: Five technology layers of the insurance platform under test

Table 1. Technology layers, validation focus, and automation risk

Layer	Technology	What automation must validate	Principal automation risk
Web	iPipeline-Resonant / Camunda new-business workflow	Process-state transitions across intake screens	Same visual element maps to different process states; locator brittleness
Mobile	Native iOS and Android app	Identical behavior on both platforms from one test design	Duplicate per-platform codebases; device fragmentation
API	ACORD TX103 XML over RabbitMQ, ELK monitoring	Message structure and field-level content	UI-centric tools treat it as an afterthought
Mainframe	Policy administration system screen sessions	Transaction entry and screen read-back via terminal emulation	Session timing and layout sensitivity under batch contention
Database	Oracle and IBM DB2	Deterministic before/after (baseline vs. candidate) table comparison	Environment drift invalidating comparisons

The research problem is therefore not "which tool automates web screens best" but whether one platform can replace the incumbent tool's coverage across all five layers at once, at lower total cost, and without a period of reduced coverage during the transition.

4. EVALUATION METHODOLOGY

4.1 Evaluation Criteria

Five criteria were derived directly from the cost and coverage problem rather than from a generic feature list:

- **C1 Breadth under one license:** coverage of all five layers without separately licensed add-on products.
- **C2 Codeless, model-based authoring:** test design usable by domain specialists without VBScript or Java skills [7].
- **C3 Self-healing object identification:** automatic adaptation to changed element attributes after a release [6], [9].
- **C4 Execution infrastructure:** cloud or grid execution instead of per-machine installed and licensed runners.
- **C5 CI/CD and ALM integration:** triggering from the delivery pipeline and reporting into test management.

4.2 Capability Coverage Matrix

Table 2 compares the incumbent tool, the candidate, and the two open-source alternatives in the author's tool background, per layer, using each vendor's or project's public documentation [1], [2], [13], [14], [15]. It records documented capability only. Whether each capability held up inside this environment was tested separately in the per-layer pilots (Section 5.4).

Table 2. Documented capability per technology layer and criterion

Layer / criterion	UFT One	accelQ	Selenium WebDriver	Playwright
Web UI	Yes	Yes	Yes	Yes
Native mobile (iOS/Android)	Via separately licensed device lab	Yes (same platform)	No; needs Appium	No; browser emulation only
API / messaging	Yes (API testing)	Yes (REST, SOAP, MQ)	No; needs extra library	HTTP only (APIRequestContext)
Mainframe terminal	Yes (Terminal Emulator	Yes (documented)	No	No

	add-in)			
Database validation	Yes (DB checkpoints)	Yes	Via custom JDBC code	Via custom code
C2 Codeless authoring	No (VBScript; record/playback)	Yes	No (code)	No (code; codegen aid)
C3 Self-healing locators	Limited (AI object ID)	Yes	No (third-party only)	No (resilient locators only)
License model	Commercial, per seat	Commercial subscription	Open source	Open source

Only accelQ and UFT documented coverage of all five layers. The open-source options were excluded for this environment because they cover neither mainframe sessions nor native mobile without assembling additional frameworks, which reintroduces the dependency overhead the organization wanted to avoid.

4.3 Weighted Scoring

The shortlisted tools were scored against C1 to C5 using the weights in Table 3. Each criterion was scored on a 1 to 5 scale , and the weighted total was the basis of the selection decision.

Table 3. Weighted evaluation scorecard

Criterion	Weight	UFT One	accelQ
C1 Breadth under one license	5	3	5
C2 Codeless authoring	5	1	4
C3 Self-healing	5	2	4
C4 Execution infrastructure	5	1	4
C5 CI/CD and ALM integration	5	1	5
Weighted total	100%	8	22

5. MIGRATION METHODOLOGY

5.1 Objective and Scope

The objective is narrow. Move production automation coverage from UFT to accelQ one layer at a time, never leave a transaction type that UFT covered without automated coverage, and declare no layer migrated until it has been shown to produce the same pass/fail outcome as UFT for the same transactions. Coverage expansion is out of scope during migration. Mixing "migrate" with "improve" makes it impossible to tell whether an outcome difference comes from the tool or from a scope change. Each of the five layers is a separate migration unit. Figure 2 shows the five phases and their gates.

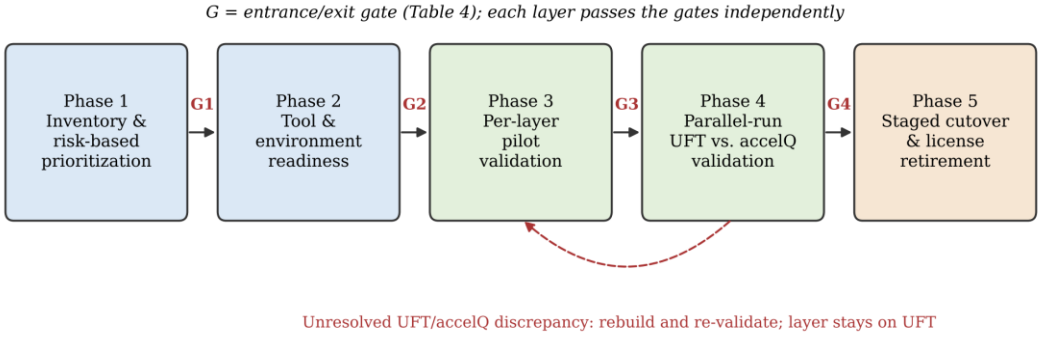


Fig 2: Five-phase, gate-controlled migration method applied independently per layer

5.2 Phase 1: Inventory and Risk-Based Prioritization

Every UFT asset is inventoried with its layer, the business transaction it validates, its execution frequency, and a risk rating describing what would go undetected if that coverage were lost. Migration order follows risk, not ease of rebuilding. The highest-risk coverage moves first, because leaving it for last maximizes the window in which a defect there could escape both tools.

5.3 Phase 2: Tool and Environment Readiness

accelQ is provisioned and connected to every environment it must reach: the web application, iOS and Android devices, the TX103 message queues, mainframe sessions through terminal emulation, and the Oracle and DB2 instances. Because execution moves from on-premises UFT machines to a cloud grid, this phase must also prove that network and security policy let the grid reach environments previously reachable only on-premises. An owner for environment stability is named, because a UFT-versus-accelQ comparison means something only if both run against the same environment state.

5.4 Phase 3: Per-Layer Pilot Validation

Each layer is piloted against a set of transaction types and pass criteria agreed in advance. Web pilots exercise process-state-dependent elements, not static pages. Mobile pilots must pass on both iOS and Android from one shared test design. A pass on only one platform counts as a partial result. API pilots validate TX103 structure and field content independently of any UI. Mainframe pilots are run repeatedly under normal batch contention, not once on a quiet system. Database pilots confirm that the established table-level baseline-versus-candidate comparison carries over unchanged. Following the outcome-report practice in [4], each pilot closes with a written record of the transaction types tested, the criteria met on first attempt, and how any gap was closed. The record's sample size is set by the technical claim being tested, not by a fixed default.

5.5 Phase 4: Parallel-Run Validation

A pilot shows that accelQ can perform a task. It does not show

that accelQ reaches the same verdict UFT would have reached. In Phase 4 the same transactions, with identical input data and the same environment state, run through both tools in a defined window, and their pass/fail outcomes are compared automatically. Every mismatch is treated as a defect and classified as (a) an accelQ asset build error, (b) a latent gap in the original UFT script, or (c) environment instability. A transaction type does not proceed until the cause is resolved. This applies the baseline-versus-candidate differential principle of the platform's full-population regression method at the tool level.

5.6 Phase 5: Staged Cutover and License Retirement

UFT seats and modules are released per layer, only after that layer completes Phase 4. A delay in one layer therefore neither forces a premature cutover elsewhere nor holds back layers that migrated cleanly, mirroring the staged retirement in [5]. Each cutover is recorded with its date, the license or module released, and the authorizing sign-off.

5.7 Governance, Roles, and Rollback

Table 4 lists the gate criteria. Three roles are kept separate: the automation builder, the environment and connectivity owner, and a domain sign-off owner per layer. This keeps the domain judgment ("is this outcome right for this product?") apart from the tool judgment ("did accelQ execute correctly?"). Because UFT coverage for a layer is released only after Phase 4, any layer not yet cut over keeps its UFT fallback. The residual risk is a gap discovered after cutover, which Phase 4's discrepancy discipline is designed to prevent.

Table 4. Phase entrance and exit gates

Phase	Entrance criterion	Exit criterion
1 Inventory	UFT asset list for all layers	Risk-ranked migration sequence agreed
2 Readiness	accelQ licensed and provisioned	Grid connectivity confirmed to all five layers
3 Pilot	Pilot transactions and pass criteria agreed per layer	Criteria met; pilot record written
4 Parallel run	Layer pilot passed	Outcomes match; all discrepancies root-caused
5 Cutover	Parallel run passed for the layer	Coverage moved; UFT seats released; cutover recorded

6. EXPERIMENTAL SETUP

6.1 Research Questions

- **RQ1 (Cost):** How did annual automation tooling cost change after UFT retirement?
- **RQ2 (Coverage parity):** Did accelQ reproduce UFT's pass/fail outcomes on each layer, and what caused the differences?
- **RQ3 (Effort):** How did test authoring, maintenance, and regression execution effort change?
- **RQ4 (Adoption):** To what extent could testers who were not scripting specialists build and maintain tests independently?

6.2 Study Context and Data Collection

The study covers the migration on the insurance platform described in Section 3,carried out by a QA team. The author led both the evaluation and the migration. Data came from license invoices and renewal quotes (RQ1), the Phase 1 inventory and the Phase 3 and Phase 4 execution logs (RQ2), time-tracking and release-cycle records (RQ3), and accelQ project authorship records (RQ4). Table 5 defines each metric.

Table 5. Metrics and data sources

Metric	Definition	Source
M1 Annual tool cost	License plus add-on and infrastructure cost per year	Invoices
M2 Assets migrated	UFT assets rebuilt in accelQ, per layer	Inventory
M3 Parity rate	Matching outcomes / parallel-run executions	Phase 4 logs
M4 Authoring time	Mean hours to build one test case	Time tracking
M5 Maintenance effort	Hours per release to repair broken tests	Time tracking
M6 Regression cycle	Elapsed time of one full regression run	Execution logs
M7 Self-heal events	Locator changes repaired automatically	accelQ reports
M8 Adoption	Testers authoring independently / team size	Authorship records

7. RESULTS AND ANALYSIS

7.1 RQ1: Cost

Table 6 compares annual tooling cost before and after migration.

Table 6. Annual automation tooling cost (or index, before = 100)

Cost component	UFT (before)	accelQ (after)	Change
Licenses / seats	50,000$	10000$	40000$
Mobile / add-on modules	N/A	Included in License	
Execution infrastructure	N/A	Included in License	
Total	50000	Included in License	

Fig 3: Annual tooling cost by component, before and after migration

7.2 RQ2: Migration Scope and Coverage Parity

Table 7 reports, per layer, the UFT assets inventoried, rebuilt in accelQ, and retired or consolidated, together with pilot and parallel-run results.

Table 7. Migration scope, pilot, and parallel-run results per layer

Layer	UFT assets inventoried	Migrated to accelQ	Retired / consolidated	Pilot test cases	Pilot first-attempt pass %
Web	300	300	0	300	95%
Mobile	N/A	60	0	60	92%

API	N/A	110	0	96	87%
Mainframe	64	64	0	64	100%
Database	80	80	0	75	90%

Fig 4: UFT assets inventoried, migrated, and retired per layer

Table 8 classifies the Phase 4 discrepancies by root cause.

Table 8. Root causes of parallel-run discrepancies

Root cause	Count	Share
(a) accelQ asset build error	6	6
(b) Latent UFT script gap	2	1
(c) Environment instability	0	0

Fig 5: Parallel-run discrepancy root causes

7.3 RQ3: Authoring, Maintenance, and Execution Effort

Table 9 compares effort measures before and after cutover.

Table 9. Effort before and after migration

Measure	UFT	accelQ	Change
M4 Authoring time per (h)	3	1	2
M5 Maintenance per release (h)	20	5	15
M6 Regression cycle (h)	20	5	15

Fig 6: Effort per measure before and after migration

8. DISCUSSION

Layer decomposition changed the decision. Evaluated on web UI alone, as most comparisons are [12], the open-source tools would have been competitive on cost. Decomposing by layer (Table 2) showed that only two tools covered the mainframe and native-mobile layers without additional frameworks. That reframed the decision from a feature contest into a total-cost and coverage question.

Parallel running pays for itself. The parallel-run phase is the most expensive step in the method, because two tools run the same work. Its value shows in Table 8:. A direct cutover would have carried those gaps forward. The finding agrees with Song's experience [5] that dual operation during transition is what makes retirement safe.

Relationship to regression methodology. Table 10 relates the method to the platform's evaluation step and its full-population regression method. All three replace an assumed equivalence with a demonstrated one, using an automated, repeatable comparison instead of a reviewer's judgment.

Table 10. Relationship to companion methods

Aspect	Tool evaluation	Migration (this paper)	Regression method
Question	Why replace UFT with accelQ?	How is it replaced layer by layer?	Did a release change behavior?
Comparison	Tools on cost and capability	UFT vs. accelQ outcome	Baseline vs. candidate code
Held constant	—	Input data, environment	Input data
Governance	Criteria C1–C5	Phase gates (Table 4)	Documented scope

9. THREATS TO VALIDITY AND LIMITATIONS

Internal validity. Effort measures before and after migration may be affected by team learning and by release content that differed between periods. Comparing the same transaction types and normalizing per test case reduces but does not remove this effect. Because the author led the migration, results are subject to observer bias. Automated outcome comparison in Phase 4 limits this for RQ2.

External validity. This is a single-organization case study. The cost result depends on this team's particular mix of seats and add-on products, and a team with a smaller mobile or API footprint could reach a different conclusion. Mainframe automation reliability depends on the specific terminal emulation and screen layout, so it should be re-piloted rather than assumed for another mainframe application.

Construct validity. Table 2 records documented capability as of 2026, which may change with product releases. The method sets no universal numeric gate thresholds. Each program must set and justify its own before Phase 3.

10. CONCLUSION AND FUTURE WORK

Licensing cost triggered the retirement of UFT, but accelQ was selected because it was the only option that covered all five layers under one license with codeless authoring. Feature-for-feature price was not the deciding factor. The migration succeeded because it was run as five independent, gated handoffs rather than one cutover. Each layer had its own pilot, its own parallel-run comparison against UFT, and its own staged license retirement Future work will apply stratified test-population selection to the migrated accelQ suite and extend the evaluation to AI-assisted test generation.

11. ACKNOWLEDGMENTS

The author thanks the QA, development, and environment teams who supported the evaluation and migration. Disclosure: an AI assistant (Claude, Anthropic) was used to restructure the manuscript into the IJCA template, edit for readability, and draw the methodology figures and chart layouts from facts and data supplied by the author. All evaluation decisions, measurements, and results are the author's own, and the author has verified the content and takes responsibility for it.

12. REFERENCES

[1] OpenText. 2024. UFT One Help Center: Terminal Emulator Add-in and API Testing. OpenText Corporation. https://admhelp.microfocus.com/uft/

[2] accelQ Inc. 2026. accelQ Platform Overview: Web, Mobile, API, Desktop and Mainframe Automation. https://www.accelq.com/product

[3] ACORD. ACORD Life & Annuity Standards: TX103 New Business Submission Transaction. ACORD Corporation.

[4] TestGrid. POC in Testing: A Complete Guide to Proof of Concept for Automation. TestGrid blog.

[5] Song, R. Cypress to Playwright Migration Guide. In Proceedings of the Pacific Northwest Software Quality Conference (PNSQC), 339–348.

[6] Martins, G., Tenório, N., and Bernardino, J. 2026. AI-Driven Software Testing: A Review. Big Data and Cognitive Computing 10, 7, Article 233. DOI: 10.3390/bdcc10070233.

[7] Gatla, G., Gatla, K., and Gatla, B. V. 2023. Codeless Test Automation for Development QA. American Scientific Research Journal for Engineering, Technology, and Sciences 91, 1, 28–35.

[8] De Reanzi, S. R. and Thangaiah, P. R. J. 2021. A survey on software test automation return on investment, in organizations predominantly from Bengaluru, India. International Journal of Engineering Business Management. DOI: 10.1177/18479790211062044.

[9] Chiranjeevi Partha Saarathy, S., Bathrachalam, S., and Rajendran, B. K. 2024. Self-Healing Test Automation Framework using AI and ML. International Journal of Strategic Management 3, 3, 45–77. DOI: 10.47604/ijsm.2843.

[10] Gojare, S., Joshi, R., and Gaigaware, D. 2015. Analysis and Design of Selenium WebDriver Automation Testing Framework. Procedia Computer Science 50, 341–346. DOI: 10.1016/j.procs.2015.04.038.

[11] García, B., Muñoz-Organero, M., Alario-Hoyos, C., and Delgado Kloos, C. 2021. Automated driver management for Selenium WebDriver. Empirical Software Engineering 26, 5, Article 107. DOI: 10.1007/s10664-021-09975-3.

[12] Shuaibu, I., Machina, M. M., and Ibrahim, M. 2019. Investigation onto the Software Testing Techniques and Tools: An Evaluation and Comparative Analysis. International Journal of Computer Applications 177, 23, 24–30. DOI: 10.5120/ijca2019919685.

[13] Selenium Project. 2026. Selenium WebDriver Documentation. https://www.selenium.dev/documentation/webdriver/

[14] Microsoft. 2026. Playwright Documentation: API Testing and Mobile Emulation. https://playwright.dev/docs/intro

[15] Appium Project. 2026. Appium Documentation. https://appium.io/docs/